ИСП
Институт Системного Программирования
им. В.П. Иванникова
Российской Академии наук

ISSN 2079-8156 (Print)
ISSN 2220-6426 (Online)

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 30, выпуск 3

Volume 30, issue 3

Москва 2018

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings/

Содержание

Table of Contents

Обнаружение ошибок, возникающих при использовании динамической памяти после её освобождения*

[2] *С.А. Асрян <asryan@ispras.ru>*
[1,3,5,6] *С.С. Гайсарян <ssg@ispras.ru>*
[1] *Ш.Ф. Курмангалеев <kursh@ispras.ru>*
[4] *А.М. Агабалян <anna.aghabalyan@ispras.ru>*
[4] *Н.Г. Овсепян <narekhnh@ispras.ru>*
[5] *С.С. Саргсян <sevaksargsyan@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Институт проблем информатики и автоматизации НАН РА, Республика Армения, Ереван, 0014, ул. П.Севака, 1*
[3] *МГУ имени М.В. Ломоносова, факультет ВМК, 2119991 ГСП-1 Москва, Ленинские горы, 2- й учебный корпус*
[4] *Ереванский государственный университет Республика Армения, г. Ереван, 0025, ул. Алека Манукяна, 1*
[5] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[6] *Национальный исследовательский университет «Высшая школа экономики» 101000, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. Существенная часть программного обеспечения написана на языках программирования C/C++. Программы на этих языках часто содержат ошибки: использования памяти после освобождения (Use After Free, UAF), переполнения буфера (Buffer Overflow) и др. В статье предложен метод обнаружения ошибок UAF, основанный на динамическом анализе. Для каждого пути выполнения программы предлагаемый метод проверяет корректность операций создания и доступа, а также освобождения динамической памяти. Поскольку применяется динамический анализ, поиск ошибок производится только в той части кода, которая была непосредственно выполнена. Используется символьное исполнение программы с применением решателей SMT (Satisfiability Modulo Theories) [12]. Это позволяет сгенерировать данные, обработка которых приводит к обнаружению нового пути выполнения.

* Работа поддержана грантом РФФИ № 17-01-00600

Ключевые слова: динамический анализ программ; покрытие кода; use-after-free.

DOI: 10.15514/ISPRAS-2018-30(3)-1

Для цитирования: Асрян С.А., Гайсарян С.С., Курмангалеев Ш.Ф., Агабалян А.М., Овсепян Н.Г., Саргсян С.С. Обнаружение ошибок, возникающих при использовании динамической памяти после освобождения. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 7-20. DOI: 10.15514/ISPRAS-2018-30(3)-1

1. Введение

В программном обеспечении могут содержаться такие ошибки, как:

- использование динамической памяти после ее освобождения (Use After Free, UAF),
- переполнение буфера или кучи (buffer/heap overflow).

Поскольку огромная часть программного обеспечения используется в критически-важных областях человеческой деятельности, наличие ошибок может привести к серьезным последствиям. Существует ряд инструментов, помогающих решить эту проблему, используя методы статического [1, 2] и динамического анализа [3, 4, 5, 6, 7].

Статический анализ предоставляет возможность исследования программного кода без его выполнения. Недостатком этого метода является отсутствие информации о состоянии программы (регистры, трасса программы, входные данные и т.д.) во время выполнения. Это приводит к большому количеству ложных срабатываний. Поэтому данный метод в большинстве случаев используется до применения динамического анализа для выявления фрагментов программы, содержащих потенциальные ошибки.

Для поиска ошибок UAF инструмент [1] выполняет анализ, похожий на анализ доступных выражений (выражение $x+y$ является доступным в точке p, если вдоль любого пути от входной точки до точки p данное выражение вычисляется, а между этими вычислениями значения x и y остаются неизменными [11]). Производится обход всех путей в программе, чтобы обеспечить выполнение условия «определение объектов до их использования». В случаи неудовлетворения данного условия считается, что было выполнено ошибочное использование памяти и выводится ошибка.

Инструмент GUEB [2] основан на исследовании бинарного кода программы. Процесс анализа разделяется на два основных этапа. На первом этапе отслеживаются операции обращения к куче и присваивания адресов для проведения анализа набора данных (какой указатель к какому элементу кучи относится). На этом этапе информация {адрес, размер} сохраняется в множествах *alloc_set* и *free_set* при создании и освобождении памяти соответственно.

На втором этапе выполняется поиск ошибок UAF. Используя собранную информацию для каждой точки программы, инструмент строит множество

access_heap, которое содержит все элементы {адрес, размер} кучи, доступные в этой точке. Если пересечение *access_heap* и *free_set* является непустом множеством, считается, что найдена ошибка UAF.

Одной из причин популярности динамического анализа является возможность исследования программ во время выполнения. Благодаря этому возможен доступ к значениям регистров и содержимому памяти. Инструмент Avalanche [3] реализует итеративный анализ исполняемого кода программы, основанный на динамической бинарной трансляции. В процессе анализа инструмент вычисляет входные данные анализируемой программы с целью автоматического обхода всех достижимых путей в программе и обнаружения аварийных завершений программы.

Инструменты DangNull [4] и FreeSentry [5] фокусируются на обнаружении и обнулении указателей на динамическую область программы после их освобождения, предотвращая появление ошибок. Оба инструмента используют статическую инструментацию программ.

Undangle [6] также предотвращает ошибки использования памяти после освобождения. Этот инструмент помечает возвращаемые значения каждой функции распределения памяти и использует анализ помеченных данных для отслеживания этих меток. Далее при освобождении памяти проверяются, какие ячейки памяти ассоциированы с соответствующей меткой, и определяются висячие указатели (указатель с ненулевым значением, ссылающийся на освобожденный область памяти).

Инструмент Mayhem [7] основан на методе поиска ошибок в бинарном коде, объединяющем офлайн- и онлайн-подходы к символьному выполнению программы. Офлайн-подход предполагает последовательное исследование путей программы: при каждом новом запуске инструмент покрывает только один путь выполнения. Недостатком является повторное выполнение общего начального фрагмента пути при каждом запуске программы. Онлайн-подход, в свою очередь, исследует все возможные пути выполнения программы одновременно, что приводит к нехватке памяти в определенный момент времени.

Объединение этих двух подходов заключается в следующем: при достижении граничного значения расхода памяти создаются контрольные точки, исследование некоторых путей останавливается с сохранением информации о текущем состоянии выполнения, контекста символьного выполнения и конкретных входных данных. После освобождения ресурсов (завершились исследование некоторых путей), восстанавливается одна из контрольных точек (с использованием сохраненных данных воспроизводится конкретное выполнение до контрольной точки). Далее выполняется загрузка контекста символьного выполнения и начинается анализ нового пути. Данный подход позволяет избежать повторного символьного выполнения программы до места создания контрольной точки.

В данной статье мы будем рассматривать подход, основанный на динамическом анализе программы с использованием динамической инструментации [8, 9]. В работе описывается метод обнаружения ошибок UAF, который проверяет корректность использования указателей для всех возможных путей выполнения программы. Этот метод основан на алгоритме покрытия кода, используемом в SAGE [10], и использует инфраструктуру динамического анализатора Triton [9].

В рамках данной работы была выполнена модификация алгоритма покрытия кода, используемого в Triton, что привело к значительному росту производительности, а также была добавлена поддержка анализа программ, работающих с файловыми входными данными, которая отсутствовала в реализации Triton.

Второй раздел статьи посвящена описанию алгоритма покрытия кода программы в Triton и предлагаемой модификации. В третьем разделе рассматривается исходная реализация обнаружения ошибок UAF и ее объединение с динамическим покрытием кода. В четвертом разделе представлены результаты.

2. Алгоритм покрытия кода

2.1 Покрытие кода в Triton

В данной статье используется алгоритм увеличения покрытия кода программы, разработанный в компании Microsoft и используемый в инструменте SAGE [10]. Этот алгоритм частично реализован в Triton. Он состоит из двух этапов:

- выбор начальных входных данных и сборка ограничений для каждого пути выполнения программы;
- получение новых входных данных с помощью решения логических выражений, состоящих из ограничений, собранных на предыдущем этапе.

Рассмотрим пример программы на рис. 1.

```
void top(char input[4]) {
  int cnt=0;
  if (input[0] == 'b') cnt++;
  if (input[1] == 'a') cnt++;
  if (input[2] == 'd') cnt++;
  if (input[3] == '!') cnt++;
  if (cnt >= 3) abort(); // error
}
```

Рис. 1. Пример программы из статьи [10]
Fig. 1. An example of a program from [10]

Чтобы можно было исследовать все пути этой программы, на вход она должна получить строку “bad!”. Чтобы получить нужные данные, алгоритм начинает свою работу, запуская программу на начальной входной строке, которая помещается в список входных данных. После первого запуска программы получается набор ограничений < $i0 \neq$ b, $i1 \neq$ a, $i2 \neq$ d, $i3 \neq$! >, где $i0$, $i1$, $i2$, $i3$ представляют ячейки памяти *input* [0], *input* [1], *input* [2] и *input* [3] соответственно.

В ходе работы алгоритма с помощью решения этих ограничений для каждого элемента из списка входных данных генерируются дочерние данные, удовлетворяющие этим ограничениям, которые далее помещаются в список входных данных. Для каждого элемента из этого списка программа заново запускается, и работа алгоритма возобновляется.

Этот процесс продолжается до тех пор, пока все элементы из списка входных данных не будут поочередно рассмотрены (псевдокод алгоритма приведен на рис. 3). Применив алгоритм для программы на рис. 1 с начальной входной строкой “good”, мы получим набор решений, представленный на рис. 2.

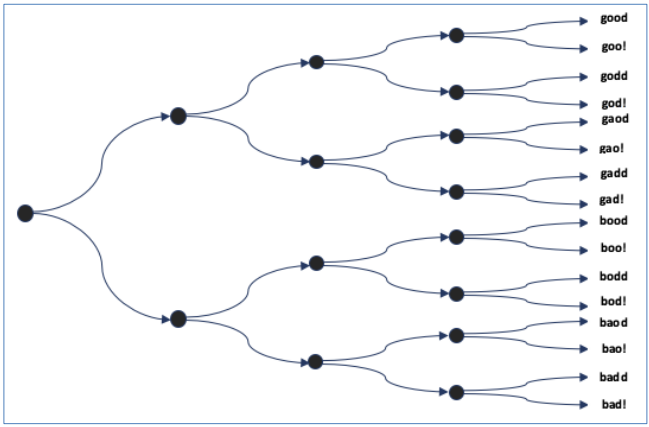


Рис. 2. Входные данные после каждой итерации алгоритма
Fig. 2. Input data after each iteration of the algorithm

```
1. runCodeCoverage(inputSeed):
2.   takeSnapshot()
3.   inputSeed.bound = 0
4.   inputList = {inputSeed}
5.   while inputList is not empty:
6.     input = getInputFromList(inputList)
7.     convertMemoryToSymbolic(input)
8.     childInputs = computeNewInputs(input)
9.     while input childInputs is not empty:
10.      inputList.append(input)
11.  if len(inputList)>0 and snapshotEnabled()
12.      restoreSnapshot()
```

```
1. computeNewInputs(input):
2.   // solve constraints using SMT solver
3.   childInputs = {}
4.   pc = ComputePathConstraint(input)
5.   for i in range(input.bound, pc.length):
6.     if (pc[0..(i-1)]) and not(pc[i]):
7.       I is solution for pc :
8.       newIn=updateWithoutOverwrite(input,I)
9.       newIn.bound = i
10.      childInputs.append(newIn)
11.   return childInputs
```

Рис. 3. Псевдокод алгоритма покрытия кода программы
Fig. 3. Pseudo code of a program code coverage algorithm

Поскольку работа данного алгоритма требует неоднократного запуска программы, в Triton реализована возможность сохранения состояния программы. Это позволяет значительно улучшить производительность выполнения. Кроме того, в Triton отсутствует часть алгоритма SAGE [10], предназначенная для уменьшения набора входных данных. Поэтому инструмент неоднократно запускает анализируемую программы на входных данных, которые не открывают новых путей. В описываемой работе в алгоритм покрытия кода был добавлен новый функционал, который позволяет достичь значительного роста производительности.

2.2 Модификация алгоритма покрытия

В оригинальной реализации алгоритма SAGE [10] в Triton после каждой итерации программа всегда получала на вход последний элемент из списка входных данных, не учитывая при этом количество открытых базовых блоков программы с помощью данного элемента. Это приводило к тому, что вместе с обработкой входных данных, которые имеют воздействие на покрытие кода программы, рассматриваются и те входные данные, с помощью которых не были открыты никакие новые пути в программе.

Поскольку в ходе работы алгоритма для каждого входного элемента генерируются ее дочерние данные, количество элементов в списке входных данных значительно увеличивается. Следовательно, для эффективного выполнения алгоритма требуется определение приоритетов для сгенерированных входных данных.

В предлагаемой модификации алгоритма каждому элементу из списка входных данных мы присваиваем вес, который представляет из себя количество базовых блоков программы, открытых этим элементом. В начале работы алгоритма входным данным присваивается нулевой вес. Во время первой итерации алгоритма подсчитываются весовые значения начальных входных данных.

После каждой итерации весовые значения обновляются следующим образом: весовые значения уже рассмотренных элементов передаются их дочерним элементам (входные данные, которые получились с помощью решения логических уравнений). Таким образом, применяется иерархический обход входных данных. Перед очередным запуском программы из списка выбирается элемент с наибольшим весом. Это позволяет значительно упростить дерево решений, как показано на рис. 4.

На рисунке видно, что после добавления весовых значений количество рассматриваемых входных данных уменьшилось почти вдвое, что в свою

очередь приводит к существенному увеличению производительности работы алгоритма (на некоторых тестах производительность выросла почти на 90%).

Еще одним недостатком Triton была поддержка программ, принимающих на вход только аргументы командной строки. Для расширения набора анализируемых программ нами была добавлена поддержка программ, использующих файлы как источник входных данных. Кроме того, была добавлена возможность определения конкретных диапазонов входных данных, которые в ходе анализа будут помечены как символьные.

Описанный подход к подсчету весовых значений не является единственным возможным, поэтому в дальнейших исследованиях будут рассматриваться и другие варианты определения этих значений.

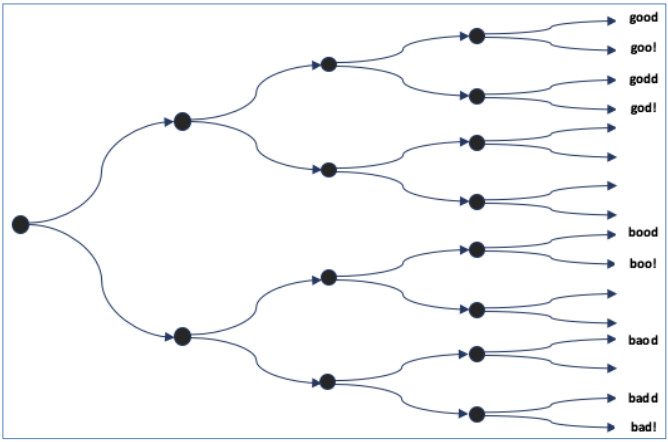


Рис. 4. Входные данные после каждой итерации алгоритма после добавления весов
Fig. 4. Input data after each iteration of the algorithm, after adding weight values

3. Поиск ошибок

Динамический анализ программы основан на исследовании программного обеспечения в процессе выполнения. Это дает возможность исследования программ с учетом определенных условий выполнения, а также позволяет использовать конкретные значения указателей. Одним из недостатков динамического анализа является требование наличия качественного покрытия кода. Однако в большинстве случаев для найденных ошибок возможна генерация входных данных, которые позволяют воспроизвести ошибку. Ошибка UAF характеризуется возникновением двух последовательных событий:

- создание висящих указателей (dangling pointers);
- доступ к памяти с использованием висящего указателя.

На рис. 5 приведён пример UAF. После проверки условия на строке 3 происходит освобождение памяти на который указывает переменная ptr (строка 4), затем управление переходит на строку 12, вследствие чего происходит повторное освобождение памяти.

3.1 Алгоритм поиска ошибок в Triton

Используя инструментацию программы, алгоритм отслеживает функции выделения (*malloc*) и освобождения (*free*) памяти. В начале работы алгоритма создаются два множества *(allocSET* и *freeSET*) для отслеживания участков памяти, которые были выделены и освобождены во время выполнения программы. Элементами данных множеств являются парами, имеющие вид («*адрес*, *размер*»).

Каждый раз при вызове *malloc/free* множества *allocSET* и *freeSET* обновляются путем добавления или удаления элементов с соответствующим адресом и размером выделенной памяти. При вызове функции *malloc* новый элемент («*адрес*_2, *размер*_2») добавляется в множество *allocSET* и удаляется из второго множества если данный элемент присутствует в множестве *freeSET* (т.е. есть совпадение как по адресу, так и по размеру).

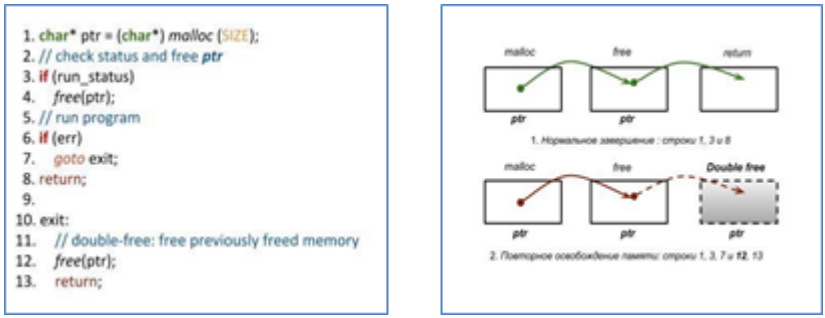


Рис. 5. Использование памяти после освобождения
Fog. 5. An example of UAF

Если данный элемент совпадает только по адресу, то перед обновлением множеств, выполняются дополнительные действия для обработки значений адреса и размера (если *размер*_2 < *размер*_1, то в *freeSET* добавляется элемент «*адрес*_2 + *размер*_2, *размер_1* - *размер*_2»). При вызове функции *free* соответствующий элемент перемещается из множества *allocSET* в множество *freeSET*.

Во время выполнения инструкций, осуществляющих доступ к памяти, проверяется наличие указателя в обоих множествах. Ошибки UAF фиксируются в двух случаях: элемент найден в множестве *freeSET* и его нет в множестве *allocSet*; один и тот же элемент встречается в *freeSET* больше одного раза. Работа алгоритма проиллюстрирована на рис. 6.

3.2 Предлагаемый метод

Для повышения эффективности поиска ошибок предлагается объединить два вышеописанных алгоритма. Объединенный метод позволяет искать ошибки UAF на разных путях программы, которые получаются из-за внедренных и нетривиальных проверок в коде. На рис. 7 приведены примеры программ, на которых обнаружение повторного освобождения памяти невозможно без использования информации о покрытии кода (из-за присутствия условных переходов, которые будут выполнены только при выполнении программы с определенными входными данными).

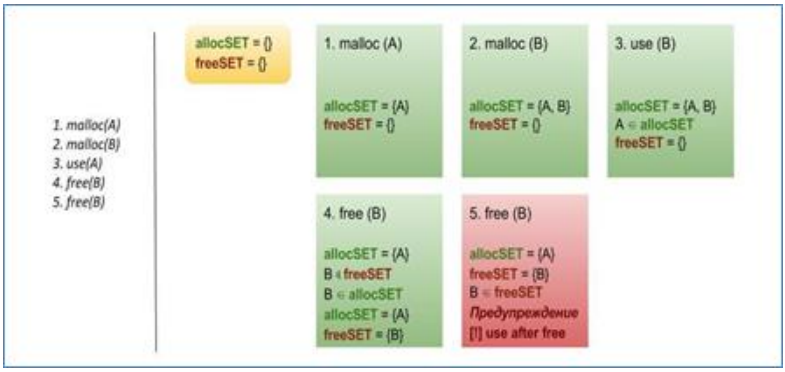


Рис. 6. Пример работы алгоритма
Fig. 6. An example of operation of the algorithm

```
1.  struct readFile {
2.    char*  buffer;
3.    int    status;
4.  };
5.
6.  int search_key_in_text(char* str, char* key) {
7.    if (key != NULL && !strstr(str, key))
8.      return -1;
9.    else
10.     return 0;
11. }
12.
13. int main(int argc, char* argv[]) {
14.   struct readFile rf;
15.   rf.buffer = (char*)malloc(sizeof(char));
16.   int fd = open(argv[1], O_RDONLY | O_CREAT);
17.   read(fd, rf.buffer, 50);
18.   printf("File cont : %s\n", rf.buffer);
19.   if (strlen(argv[2]) > 64 && !checkAttr(argv[2])) {
20.     rf.status = 11;
21.     free(rf.buffer);
22.   }
23.   if (!search_key_in_text(rf.buffer, argv[2])) {
24.     // do some stuff
25.   }
26.   free(rf.buffer); // Use after free
27. }
```

```
1.  int myatoi(char *p) {
2.    int i = 0;
3.    while (*p != '\x00' && *p >= '0' && *p <= '9') {
4.      i *= 10;
5.      i += *(unsigned char *)p - '0';
6.      p++;
7.    }
8.    return i;
9.  }
10.
11. int main(int argc, char *argv[]) {
12.   char* str = (char*) malloc (SIZE);
13.   int num;
14.   if (argc != 2) {
15.     printf("Usage: %s <string>\n", argv[0]);
15.     exit(-1);
17.   }
18.   strcpy(str, argv[1]);
19.
20.   num = myatoi(str);
21.   if (num >= 33 && ! (num % 2)) {
22.     printf("ok\n");
23.     free(str)
24.   }
25.   if (num > 1024 ) {
26.     printf("Number is out of range\n");
27.     free(str); // Use after free
28.   }
29.   return 0;
30. }
```

*Рис. 7. Ошибки повторного освобождения памяти. Первый пример основан на ошибке UAF в программе libssh. На первом примере ошибка может возникнуть на строке **26**, а на втором примере на строке **27***
Fig. 7. Errors of double memory freeing. The first example is based on the UAF error in the libssh program. In the first example, an error may occur on line 26, and in the second example on line 27

Для примеров, приведенных на рис. 7, ошибка UAF произойдет, если выполнение программ достигнет строк 21 и 23 для первой и второй программы соответственно. Использование предлагаемого метода позволяет найти входные данные для достижения нужного блока в коде (строки 20-21 и 22-23, рис. 7) и потом проверить данную часть на наличие ошибок UAF.

3.3 Сравнение подходов динамического анализа

В табл. 1 подход, описываемый в данной статье, сравнивается в подходами, применяемыми в Mayhem и Triton.

Табл. 1. Результаты сравнения
Table 1. Results of comparison

	Предлагаемый подход	**Mayhem**	**Triton UAF**
Использование покрытия кода	+	+	-
Офлайн-подход символьного выполнения	+	+	+
Онлайн-подход символьного выполнения	-	+	-
Приоритеты обрабатываемых входных данных	+	+	-
Поддержка файлов, в качестве входных данных	+	+	-

4. Результаты

Предлагаемый метод был протестирован на синтетических тестах, в том числе и на приведённых примерах (рис. 1, 7), результаты тестирования приведены на рис. 8. Данные результаты показывают, что на синтетических тестах, по сравнению с реализацией Triton, производительность выросла примерно на 80%. Запуск анализа на реальных программах показал, что в большинстве случаев количество символьных уравнений становится настолько большим, что алгоритм покрытия кода Triton не может решить полученные уравнения для всех путей.

Для проверки предложенного подхода нами были специально внесены ошибки UAF в исходный код реальных проектов. Были исследованы проекты: gvgen из пакета graphviz, jasper из пакета libjasper-runtime и gif2rgb из пакета giflib. На данных проектах ошибки были найдены на различных уровнях встраивания. В случае программы gvgen встраивание было выполнено за пределами одной функции, максимальная глубина составляло три уровня (функции). Код встраивания в данном случае представлял с собой условное выражение, связанное с входными данными, и код самой ошибки. Выполнение этого условия приводило к воспроизведению данной ошибки.

В проектах jasper и gif2rgb из-за сложности полученных символьных уравнений встраивание было выполнено в пределах только одной функции. Код встраивания был непосредственно кодом ошибки. Также были выделены отдельные участки кода из реальных программ содержащие UAF на которых данный подход смог найти соответствующие ошибки.

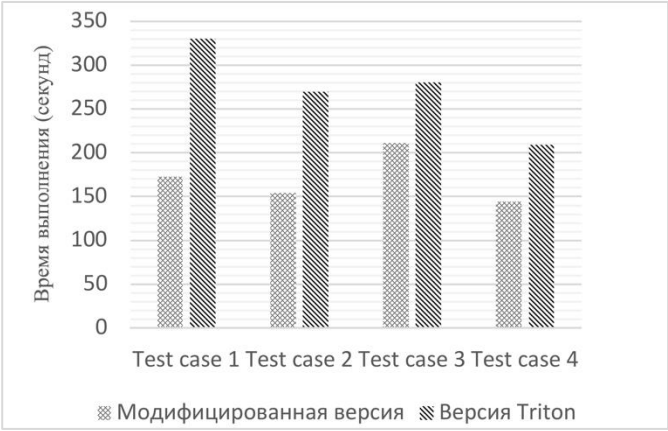


Рис. 8. Результаты сравнения относительно времени выполнения анализа
Fig. 8. Comparison results relative to the analysis execution time

5. Заключение

В статье представлен метод обнаружения ошибок UAF, возникающих при неправильной обработке указателей на динамическую память. Метод реализован с помощью инфраструктуры Triton [9] на базе алгоритма [10] и алгоритма обнаружения ошибок UAF. После проведенных модификаций и улучшений существующей реализации был получен прирост производительности выполнения анализа.

Список литературы

[1]. D. Dewey, B. Reaves, P. Trainor. Uncovering Use-After-Free Conditions in Compiled Code. In Proc of the 10th International Conference on Availability, Reliability and Security (ARES), 2015, pp. 90-99

[2]. J. Feist, L. Mounier, ML. Potet. Statically detecting use after free on binary code. Journal of Computer Virology and Hacking Techniques, vol. 10, issue 3, 2014, pp 211-217

[3]. И.К. Исаев, Д.В. Сидоров, А.Ю. Герасимов, М.К. Ермаков. Avalanche: Применение динамического анализа для автоматического обнаружения ошибок в программах, использующих сетевые сокеты. Труды ИСП РАН, том 21, 2011 г., стр. 55-70.

[4]. B. Lee, Ch. Song, Y. Jang, T. Wang. Preventing Use-after-free with Dangling Pointers Nullification. In Proc of the Network and Distributed System Security Symposium, 2015, https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/preventing-use-after-free-dangling-pointers-nullification/, дата обращения 05.05.2018

[5]. Yves Younan. FreeSentry: Protecting Against Use-After-Free Vulnerabilities Due to Dangling Pointers. In Proc of the Network and Distributed System Security Symposium, 2015, https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/freesentry-protecting-against-use-after-free-vulnerabilities-due-dangling-pointers/, дата обращения 05.05.2018

[6]. J. Caballero, G. Grieco, M. Marron, A. Nappa. Undangle: Early Detection of Dangling Pointers in Use-After-Free and Double-Free Vulnerabilities. In Proceedings of the 2012 International Symposium on Software Testing and Analysis, 2012, pp. 133-143

[7]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley. Unleashing MAYHEM on Binary Code. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, 2012, pp. 380-394

[8]. Pin – A Dynamic Binary Instrumentation Tool, https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool, дата обращения 05.05.2018

[9]. Triton – Dynamic Binary Analysis Framework, https://triton.quarkslab.com/, дата обращения 05.05.2018

[10]. P. Godefroid, M. Y. Levin, D. Molnar. Automated Whitebox Fuzz Testing. In Proceedings of NDSS'2008 (Network and Distributed Systems Security), 2008, pp. 151-166.

[11]. A. Aho, J. Ullman, R. Sethi, M. S. Lam. Compilers: Principles, Techniques, and Tools. Addison Wesley; 2nd edition, September 10, 2006, 1000 p.

[12]. Leonardo de Moura, Nikolaj Bjørner. Z3: an efficient SMT solver. In Proceedings of the 14th international conference on Tools and algorithms for the construction and analysis of systems, 2008, pp. 337-340

Асрян С.А., Гайсарян С.С., Курмангалеев Ш.Ф., Агабалян А.М., Овсепян Н.Г., Саргсян С.С. Обнаружение ошибок, возникающих при использовании динамической памяти после освобождения. *Труды ИСП РАН*, том 30, вып. 3, 2018 г., стр. 7-20

Dynamic detection of Use After Free bugs

[2] *S.A Asryan <asryan@ispras.ru>*
[1, 3, 5, 6] *S.S. Gaissaryan <ssg@ispras.ru>*
[1] *Sh. F. Kurmangaleev <kursh@ispras.ru>*
[4] *A.M. Aghabalyan <anna.aghabalyan @ispras.ru>*
[4] *N.H. Hovsepyan <narekhnh@ispras.ru>*
[4] *S.S Sargsyan <sevaksargsyan@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Institute for Informatics and Automation Problems of NAS RA, 1, P. Sevak str., Yerevan, 0014, Republic of Armenia,*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[4] *Yerevan State University 1, Alex Manoogian str., Yerevan, 0025, Republic of Armenia*
[5] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia*
[6] *National Research University Higher School of Economics (HSE) 11 Myasnitskaya str., Moscow, 101000, Russia*

Abstract. The article describes new method of use after free bug detection using program dynamic analysis. In memory-unsafe programming languages such as C/C++ this class of bugs mainly accurse when program tries to access specific area of dynamically allocated memory that has been already freed. This method is based on combination of two basic components. The first component tracks all memory operations through dynamic binary instrumentation and searches for inappropriate memory access. It preserves two sets of memory address for all allocation and free instructions. Using both sets this component checks whether current memory is accessible through its address or it has been already freed. It is based on dynamic symbolic execution and code coverage algorithm. It is used to maximize the number of execution paths of the program. Using initial input, it starts symbolic execution of the target program and gathers input constraints from conditional statements. The new inputs are generated by systematically solving saved constraints using constraint solver and then sorted by number of basic blocks they cover. Proposed method detects use after free bugs by applying first component each time when second one was able to open new path of the program. It was tested on our synthetic tests that were created based on well-known use after free bug patterns. The method was also tested on couple of real projects by injecting bugs on different levels of execution.

Keywords: program dynamic analysis; use after free bug; dynamic symbolic execution; code coverage; instrumentation.

DOI: 10.15514/ISPRAS-2018-30(3)-1

For citation: Asryan S.A., Gaissaryan S.S., Kurmangaleev Sh. F., Aghabalyan A.M., Hovsepyan N.H., Sargsyan S.S. Dynamic detection of Use After Free bugs. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018. pp. 7-20 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-1

References

[1]. D. Dewey, B. Reaves, P. Trainor. Uncovering Use-After-Free Conditions in Compiled Code. In Proc of the 10th International Conference on Availability, Reliability and Security (ARES), 2015, pp. 90-99

[2]. J. Feist, L. Mounier, ML. Potet. Statically detecting use after free on binary code. Journal of Computer Virology and Hacking Techniques, vol. 10, issue 3, 2014, pp 211-217

[3]. Ildar Isaev, Denis Sidorov, Alexander Gerasimov, Mikhail Ermakov. Avalanche: Using dynamic analysis for automatic defect detection in programs based on network sockets. Trudy ISP RAN/Proc. ISP RAS, vol. 21, 2011, pp. 55-70 (in Russian).

[4]. B. Lee, Ch. Song, Y. Jang, T. Wang. Preventing Use-after-free with Dangling Pointers Nullification. In Proc of the Network and Distributed System Security Symposium, 2015, https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/preventing-use-after-free-dangling-pointers-nullification/, accessed at 05.05.2018

[5]. Yves Younan. FreeSentry: Protecting Against Use-After-Free Vulnerabilities Due to Dangling Pointers. In Proc of the Network and Distributed System Security Symposium, 2015, https://www.ndss-symposium.org/ndss2015/ndss-2015-programme/freesentry-protecting-against-use-after-free-vulnerabilities-due-dangling-pointers/, accessed at 05.05.2018

[6]. J. Caballero, G. Grieco, M. Marron, A. Nappa. Undangle: Early Detection of Dangling Pointers in Use-After-Free and Double-Free Vulnerabilities. In Proceedings of the 2012 International Symposium on Software Testing and Analysis, 2012, pp. 133-143

[7]. Sang Kil Cha, Thanassis Avgerinos, Alexandre Rebert and David Brumley. Unleashing MAYHEM on Binary Code. In Proceedings of the 2012 IEEE Symposium on Security and Privacy, 2012, pp. 380-394

[8]. Pin – A Dynamic Binary Instrumentation Tool, https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool, accessed at 05.05.2018

[9]. Triton – Dynamic Binary Analysis Framework, https://triton.quarkslab.com/, accessed at 05.05.2018

[10]. P. Godefroid, M. Y. Levin, D. Molnar. Automated Whitebox Fuzz Testing. In Proceedings of NDSS'2008 (Network and Distributed Systems Security), 2008, pp. 151-166.

[11]. A. Aho, J. Ullman, R. Sethi, M. S. Lam. Compilers: Principles, Techniques, and Tools. Addison Wesley; 2nd edition, September 10, 2006, 1000 p.

[12]. Leonardo de Moura, Nikolaj Bjørner. Z3: an efficient SMT solver. In Proceedings of the 14th international conference on Tools and algorithms for the construction and analysis of systems, 2008, pp. 337-340

Buffer Overflow Detection via Static Analysis: Expectations vs. Reality

I.A. Dudina <eupharina@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.

Abstract. Over the last few decades buffer overflow remains one of the main sources of program errors and vulnerabilities. Among other solutions several static analysis techniques were developed to mitigate such program defects. We analyzed different approaches and tools that address this issue to discern common practices and types of detected errors. Also, we explored some popular sets of synthetic tests (Juliet Test Suite, Toyota ITC benchmark) and set of buggy code snippets extracted from real applications to define types of defects that a static analyzer is expected to uncover. Both sources are essential to understand the design goals of a production quality static analyzer. Test suites expose a set of features to support that is easy to understand, classify, and check. On the other hand, they don't provide a real picture of a production code. Inspecting vulnerabilities is useful but provides an exploitation-biased sample. Besides, it does not include defects eliminated during the development process (probably with the help of some static analyzer). Our research has shown that interprocedural analysis, path-sensitivity and loop handling are essential. An analysis can really benefit from tracking affine relations between variables and modeling C-style strings as a very important case of buffers. Our goal is to use this knowledge to enhance our own buffer overrun detector. Now it can perform interprocedural context- and path-sensitive analysis to detect buffer overflow mainly for static and stack objects with approximately 65% true positive ratio. We think that promising directions are improving string manipulations handling and combining taint analysis with our approaches.

Keywords: software error detection; static analysis; buffer overrun

DOI: 10.15514/ISPRAS-2018-30(3)-2

For citation: Dudina I.A. Buffer Overflow Detection via Static Analysis: Expectations vs. Reality. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 21-30. DOI: 10.15514/ISPRAS-2018-30(3)-2

1. Introduction

Buffer overflow is a type of program defect caused by buffer access with index that exceeds buffer's bounds. This can lead to a program crash or even to a security vulnerability. Defects of such kind are still common, despite all efforts made to

eliminate them. There are several techniques one can apply to detect buffer overflows. One approach is to employ testing and dynamic analysis. These methods don't suffer from false positives, but in most cases, it's impossible to check all execution paths, so some defects can remain undetected. Another approach is to analyze program code without executing it. In this way, one can find a defect on any path, even rarely executed. In this paper, we will focus on the latter approach known as static analysis.

We are interested in building a buffer overflow detector that is applicable to large C/C++ programs with millions of lines of code while producing decent analysis performance and quality. Basic properties of the algorithms constituting such a detector are well-known and include among others interprocedural analysis, path sensitivity, and loop handling. However, after initial support for these features has been made and the quality goals achieved, it is unclear which direction to choose for the further improvement. The usual development pace that comes from the customer feedback and own code analysis may be not enough. In the following chapters, we'll overview possible sources of inspiration for the buffer overflow detector development, present our short survey that is based on the buffer overflow-related vulnerabilities sample from the CVE database, then briefly describe our experience of developing an overrun detector as a part of the Svace tool, and present our conclusions from tools and vulnerabilities analysis.

2. Buffer overflow detection techniques and tools

There exist many static analysis tools that can detect buffer overflows. In this section, we conduct a brief survey on the most popular methods.

Some buffer overflows can be detected during the process of lexical analysis, like in the ITS4 tool [1]. Most common errors and bad patterns can be found at this level. This technique can work really fast and, as it doesn't involve compilation, can be easily applied to any code, even if it is not complete. As a result, such analysis can be performed "on-the-fly" during the process of code development with IDE, so that erroneous patterns are eliminated on the very early coding stage. Of course, such a lightweight method is far from being sound, i.e. it misses many defects. Even changing the name of a variable can prevent such tools from detecting a defect.

To detect more defects a deeper analysis of code is needed. To achieve this, many tools use the idea of abstract interpretation [2]. Some tools chose different numerical abstract domains to implement the analysis of integer index values, buffer sizes, and string lengths. These domains include intervals, zones, octagons, affine equalities, interval linear equalities, convex polyhedra, tropical polyhedra, etc. [3]. Tools based on these approaches derive sound relationship between integer values listed above in varying degrees of precision. Soundness is a major advantage of such tools, but less precise domains produce large number of false positives, while analysis with more precise domains doesn't scale on many real-world programs.

Another popular approach is symbolic execution. The main idea of this method is performing analysis by traversing all paths in a function separately. This approach

can be used to build a path-sensitive detector i.e. that can find errors that, at the same time, occur only on a certain feasible function path and are not inevitable for any single point from this path alone. While processing a particular path, the analyzer keeps track of variables values and relationships and computes a path predicate, i.e. a conjunction of all corresponding branch conditions that are taken along this path. This information is used to prune infeasible paths and check buffer access instructions. Analyzing all paths in a function can be a challenging task due to the path explosion, so a number of techniques are proposed to reduce this problem. A simple, but often effective approach is to abandon the idea of full path coverage and just to stop the analysis after some threshold or time limit reached. Another approach is to merge symbolic states at join points, preserving path-sensitivity of analysis by providing guard conditions for joined states. Third approach, first introduced in Marple, is employing demand-driven analysis [4], [5], i.e. reducing the set of analyzed paths by focusing only on those that end with buffers access.

One of the main obstacles for all mentioned symbolic execution-based approaches is handling loops. Typical solution is to implement some heuristics to handle the most simple and common loops and ignore other loops. However, there are methods proposed to handle loops with multiple paths inside and summarize their effect on program values [6].

Many buffer overflow errors are caused by violations of function contracts. This can happen when a caller of a library or a user function provides unexpected data to a function, or, on the contrary, a function is not able to correctly handle all input cases implied by the contract. Interprocedural analysis is needed to detect such inconsistencies.

On the lexical analysis level, formal and actual arguments matching can be based on similar variables names and usually happens only for the well-known library callees like `memcpy`. For more rigorous scan some tools analyze the whole program as a unified inter-procedural graph. The monomorphic analysis merges information for every call-site — efficient, but imprecise approach. The polymorphic analysis treats each call site individually, so this approach provides context-sensitivity but scales poorly.

An alternative approach is using some approximation of a function’s behavior when analyzing its caller. These approximations can be provided in user’s annotations, but they are not always available. A tool can use its own findings obtained by the callee analysis as an approximation. This approached is called summary-based. By choosing the right function order, a tool can minimize the number of missing summaries, but handling recursion still requires additional tricks, e.g. making several analysis passes over strongly connected components of the call graph.

3. Buffer overflow detection tools benchmarking

For the past twenty years several studies have been published on evaluating and testing buffer overflow detectors. In addition, there exist different test suites, which

provide sets of synthetic buggy and correct code snippets to test the abilities and false positive rate of static analysis tools.

One of the biggest and probably the most popular benchmark is Juliet Test Suite C/C++, created by NSA's Center for Assured Software (CAS) [7]. For C/C++ code it contains 64,099 test cases tagged by CWE entries. Groups corresponding to buffer overflow defects are CWE 121 — "Stackbased Buffer Overflow" (4,968 tests), CWE 122 — "Heapbased Buffer Overflow" (5,922 tests), CWE 124 — "Buffer Underwrite" (2,048 tests), CWE 126 — "Buffer Over-read" (1,452 tests), and CWE 127 — "Buffer Under-read" (2048 tests). Tests in this suite are also tagged with a number called "flow variant" that represents the complexity of control and data flow in a particular test case.

Control flow variants cover different types of conditionals (e.g. `STATIC_CONST_FIVE==5`, `globalReturnsTrueOrFalse()`, etc.) and different control statements (`switch`, `while`, etc.). Data flow variants describe many types of intraprocedural data flow and interprocedural interaction, e.g. data passing through function arguments (via pointer, C++ reference, array, container, etc.), return value, global variable, etc. There are many flow variants that represent C++-specific features and not applicable to C-tests.

We noticed that the distribution of the flow variants is close to uniform in groups of our interest. Another observation is large number of tests involving wide characters. Many tests contain library function usage, e.g. `memcpy`-like functions, string manipulations, format string processing, etc.

Toyota ITC Benchmark is a test suite created by Toyota InfoTechnology Center aimed at the static analysis tool evaluation [8]. It contains 1,276 simple tests (638 erroneous and 638 correct) divided into 9 types and 51 sub-types. Our interest is in the following tests: sub-types "static buffer overrun" (54 cases), "static buffer underrun" (13 cases) from the "static memory" type and sub-types "dynamic buffer overflow" (32 cases), "dynamic buffer underrun" (39 cases) from the "dynamic memory" type. Each case is represented by a pair of a buggy test and a fixed test.

These samples cover following features in varying combinations: (i) static, stack and heap buffers; (ii) different element types (`char`, `int`, `float`, `struct`, etc.); (iii) index calculations (constant, linear and non-linear expressions, passed as an argument or returned from a function, loop variables and array elements); (iv) obtaining buffer address (local/global variable, function argument, pointer arithmetic including loop variables and aliases); (v) buffer size (heap buffers only with constant sizes, pointer casting); (vi) access types (via index, pointer dereference, in a library function, in a string function).

4. Survey on overflow-related CVEs

We believe that although evaluating with a test suite could give a good insight in a particular tool's abilities, any test suite alone cannot perfectly represent the whole populations of buffer overflow defects in real code. One (but not the only one) noble

goal for static analyzers is to prevent security vulnerabilities to sneak in the project source code. We wanted a better understanding of the features of a static analyzer that are more or less important for achieving this goal. Our survey technique was inspired by [9] and we mostly followed in their footsteps to produce a set of vulnerabilities to classify.

We have to note that detection of exploitable vulnerabilities is not the only goal of a static analyzer. Still there are some types of defects that don't lead to vulnerabilities or may not be exploited with ease, but it is undesirable to have those in the source code. Besides, we believe that nowadays developers more intensively use different (static and/or dynamic) analysis tools before releasing the product. For this reason, many simple defects are eliminated during the development process and don't appear in the vulnerability databases. Consequently, we think that analysis of the vulnerabilities can reveal the weakest sides of modern static analysis and show potential improvement directions.

First of all, we have randomly picked 100 entries from the "overflow" category from the CVE database [10]. For 25 of them we could find a source code of the vulnerable version to inspect. For each defect, we have studied its causes in the code and then classified the defect by several attributes. Our set of attributes is based on the taxonomy provided in [11] with some changes.

Our first insight is that there are some trends in our sample that can be explained by the source of this sample (vulnerability database): (i) most of the overflows from our sample (72%) happened on write memory access, only few on read access; (ii) only the upper bounds of buffers are exceeded in the defects from our sample; (iii) almost all defects (92%) occurred when tainted data (unbounded data from network, file read, input parameters etc.) overflowed some buffer.

We also noticed that simple errors (e.g. using unsafe functions like `strcpy`) are present in the old code (before 2010), but rarely in the late entries. We believe that this can be partially explained by the usage of code analysis tools.

In our sample about a half of overflowed buffers (48%) reside on a stack, other half (48%) is allocated on a heap, and just a few are global variables.

40% of all defects have overflowed buffer accessed via index (e.g. `buf[i]`), 12% via pointer dereference, 44% via library calls, 24% of which are string functions. The latter requires C-strings modeling to properly analyze such patterns. When buffer is accessed in a library call, we think of size/limit argument as an index (when it's reasonable) for further investigation.

According to our data, 48% of all vulnerable buffers have constant size (all stack and static buffers and a few buffers on the heap). Another 16% have a size that is calculated as a linear combination of other variables. As a result, almost half of all inspected defects require deep analysis of integer variables relationship to detect them.

Another feature that we have evaluated for every entry is whether buffer allocation is global or resides in the same function with buffer access. We have found that this

is true only for 24% of defects. On the other hand, all index calculations are in the same function with the access in 32% of defects. Both properties are true for 12% of defects. It follows from the foregoing that interprocedural analysis is essential for buffer overflow detection.

Last thing that we have checked is whether there exists a program point that any path through this point will lead to a corresponding error. If there is no such point, then we assume that path-sensitive analysis is needed to detect this defect. Our sample contains only 28% of defects, for which such a program point exists. This means that path-sensitivity will provide the real advantage for a static analysis tool.

5. Svace buffer overrun detector

Svace is a static analysis tool that is designed to find as many defects of different types as possible with few false positives and acceptable analysis time [12]. The purpose of this work is to improve the Svace buffer overflow detector with the most needed features. Our detector implements the interprocedural path-sensitive detection algorithm based on symbolic execution with state merging [13]. For now, the analysis scope is limited to detection overflows of buffers with compile-time-known size. Our detector looks for faulty paths in a function, i.e. it reports a warning if it finds a path that for any input values is either infeasible or produces an error. Such a strict defect definition is chosen to prevent many false positives caused by unknown function preconditions.

For a buffer access instruction, we collect a predicate that implies that there exists a faulty path through this instruction. We use an SMT solver to search a solution for this predicate if any. In case of this formula is satisfiable, we use its model provided by the solver to extract a faulty path. It follows from our experience that simply asking solver for any index value that exceeds buffer bounds in our case leads to many false positives. Reasons for that are unknown function precondition and symbolic path conditions being not precise enough (due to poor loop handling, calls of unknown or complex functions, etc.).

Our interprocedural analysis is implemented using summaries. In the function summary, we save the information about relationships between integer values on function entry and exit points. We also save overflow conditions for those input-dependent buffer accesses whose correctness can only be checked in the caller context. Such facts can be propagated to the caller more than once, so the analysis can find an overflow of a buffer allocated in a function that is far away on the call stack from a function with the access instruction. We also implemented a heuristic to handle simple loops that have an inductive variable iterating over an arithmetic progression. Currently on Android 5.0.2 our detector emits 351 warnings with 65% true-positive ratio.

6. Conclusion

We have inspected a number of buffer overflow test suites, related CVE entries, and the source code of large production projects that our tool regularly analyzes. All three sources are essential to understand the design goals of a production quality static analyzer. Test suites expose a set of features to support that is easy to understand, classify, and check. On the other hand, they don't provide a real picture of a production code. Inspecting vulnerabilities is useful but provides an exploitation-biased sample. Besides, it does not include defects eliminated during the development process (probably with the help of some static analyzer). Finally, while developing a static analyzer one always deals with false positives produced by the tool and reported by customers, but getting false negative samples is much more difficult. True positives reported by the other tools could be useful, but most of the state-of-the-art tools are proprietary and their results are closed.

From what has been said above it follows that interprocedural analysis, path-sensitivity and loop handling are essential. An analysis can really benefit from tracking affine relations between variables and modeling C-style strings as a very important case of buffers.

Our current goal is to improve the Svace buffer overflow detector to reduce the number of false negatives while preserving the moderate level of false positives. For the aforementioned reasons, we think that the most promising directions are handling buffers with dynamic size, C-string modeling, and tracking tainted values. We are working now on the extension of our detection technique described in Section 5 by tracking string length changes happening during string operations in much the same way as we track buffer indexes while calculating integer values. We believe that this will be sufficient for most of cases, but there are some promising works in the area of string solvers [14] that would additionally allow to track also string contents.

As we have seen, static analysis detection of buffer overflows requires a number of techniques from vastly various fields to move on the road from expectations to real code, and there will always be a way to go.

References

[1]. J. Viega, J. T. Bloch, Y. Kohno, and G. McGraw. Its4: A static vulnerability scanner for c and c++ code. In Proceedings of the 16th Annual Computer Security Applications Conference, 2000, pp. 257-269.

[2]. P. Cousot and R. Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977, pp. 238–252.

[3]. X. Allamigeon. Static analysis of memory manipulations by abstract interpretation – Algorithmics of tropical polyhedra, and application to abstract interpretation. PhD thesis, Ecole Polytechnique X, Nov. 2009. [Online]. Available: https://pastel.archives-ouvertes.fr/pastel-00005850, accessed: 2018-04-08.

[4]. W. Le and M. L. Soffa. Marple: A Demand-Driven Path-Sensitive Buffer Overflow Detector. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008, p. 272-282.
[5]. L. Li, C. Cifuentes, and N. Keynes. Practical and effective symbolic analysis for buffer overflow detection. In Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, pp. 317– 326.
[6]. X. Xie, Y. Liu, W. Le, X. Li, and H. Chen. S-looper: automatic summarization for multipath string loops. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, 2015, pp. 188–198.
[7]. Juliet Test Suite v1.2 for C/C++. User Guide. Available: https://samate.nist.gov/SRD/around.php#juliet_documents, accessed: 2018-04-08.
[8]. S. Shiraishi, V. Mohan, and H. Marimuthu. Test suites for benchmarks of static analysis tools. In Proceedings of the 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Nov 2015, pp. 12–15.
[9]. T. Ye, L. Zhang, L. Wang, and X. Li. An Empirical Study on Detecting and Fixing Buffer Overflow Bugs. In Proceedings of the 2016 IEEE International Conference on Software Testing, Verification and Validation, 2016, pp. 91–101.
[10]. CVE security vulnerability database. Security vulnerabilities, exploits, references and more. Available: https://www.cvedetails.com/index.php, accessed: 2018-04-08.
[11]. K. Kratkiewicz and R. Lippmann. A taxonomy of buffer overflows for evaluating static and dynamic software testing tools. In Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics, vol. 500, 2006, pp. 44-51.
[12]. A. Borodin and A. Belevantcev. A static analysis tool Svace as a collection of analyzers with various complexity levels. Trudy ISP RAN /Proc. ISP RAS, vol. 27, issue 6, 2015, pp. 111–134.
[13]. I.A. Dudina and A.A. Belevantsev. Using static symbolic execution to detect buffer overflows. Programming and Computer Software, vol. 43, no. 5, 2017, pp. 277–288. DOI: 10.1134/S0361768817050024.
[14]. Y. Zheng, X. Zhang, and V. Ganesh. Z3-str: A z3-based string solver for web application analysis. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 114–124.

Статический анализ для поиска переполнения буфера: актуальные направления развития

И.А. Дудина <eupharina@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
Московский государственный университет им. М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. В последние десятилетия переполнение буфера остаётся одним из главных источников программных ошибок и эксплуатируемых уязвимостей. Среди прочих подходов к устранению подобных дефектов активное развитие получили различные методы статического анализа. В работе рассматриваются основные подходы и инструменты, используемые для решения этой задачи, с целью выявить наиболее популярные методы и типы обнаруживаемых ошибок. Также исследованы наборы

синтетических тестов (Juliet Test Suite, Toyota ITC benchmark) и выборка фрагментов кода реальных приложений, содержащих эксплуатируемую ошибку переполнения буфера. Для понимания направлений развития промышленного статического анализатора важно рассматривать оба эти источника примеров ошибочных программ. Наборы тестов очерчивают круг ситуаций, которые необходимо поддержать в анализаторе, при этом их легко понять, классифицировать и проверить. С другой стороны, они не отражают распределение таких ситуаций в реальном коде. Выборка уязвимостей из промышленных проектов также представляет интерес для исследования, но оказывается смещённой в сторону эксплуатируемых ошибок и к тому же не включает ошибки, исправленные на стадии разработки (возможно, как раз с использованием статического анализатора). Полученные данные были использованы для выделения основных шаблонов дефектов, которые должен обнаруживать статической анализатор с точки зрения пользователя. В результате исследования к наиболее важным возможностям статического анализатора были отнесены межпроцедурный путе- и контекстно-чувствительный анализ, а также базовая поддержка циклов. Кроме того, полезными оказываются отслеживание аффинных отношений между переменными и моделирование строк как важного случая использования массивов. Результаты данного исследования используются для улучшения детектора переполнения буфера, реализованного в рамках инфраструктуры статического анализатора Svace. На данный момент используется межпроцедурный чувствительный к путям и контексту анализ, позволяющий обнаруживать переполнения буфера на стеке и в статической памяти с долей истинных срабатываний 65%. По результатам исследования наиболее перспективными направлениями представляются поддержка строковых операций и внедрение анализа помеченных данных в имеющиеся подходы.

Ключевые слова: анализ программ; статический анализ; переполнение буфера

DOI: 10.15514/ISPRAS-2018-30(3)-2

Для цитирования: Дудина И.А. Статический анализ для поиска переполнения буфера: актуальные направления развития. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 21-30 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-2

Список литературы

[1]. J. Viega, J. T. Bloch, Y. Kohno, and G. McGraw. Its4: A static vulnerability scanner for c and c++ code. In Proceedings of the 16th Annual Computer Security Applications Conference, 2000, pp. 257-269.

[2]. P. Cousot and R. Cousot. Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977, pp. 238–252.

[3]. X. Allamigeon. Static analysis of memory manipulations by abstract interpretation – Algorithmics of tropical polyhedra, and application to abstract interpretation. PhD thesis, Ecole Polytechnique X, Nov. 2009. [Online]. Available: https://pastel.archives-ouvertes.fr/pastel-00005850, accessed: 2018-04-08.

[4]. W. Le and M. L. Soffa. Marple: A Demand-Driven Path-Sensitive Buffer Overflow Detector. In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, 2008, p. 272-282.

[5]. L. Li, C. Cifuentes, and N. Keynes. Practical and effective symbolic analysis for buffer overflow detection. In Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2010, pp. 317– 326.

[6]. X. Xie, Y. Liu, W. Le, X. Li, and H. Chen. S-looper: automatic summarization for multipath string loops. In Proceedings of the 2015 International Symposium on Software Testing and Analysis, 2015, pp. 188–198.

[7]. Juliet Test Suite v1.2 for C/C++. User Guide. Режим доступа: https://samate.nist.gov/SRD/around.php#juliet_documents, дата обращения: 2018-04-08.

[8]. S. Shiraishi, V. Mohan, and H. Marimuthu. Test suites for benchmarks of static analysis tools. In Proceedings of the 2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Nov 2015, pp. 12–15.

[9]. T. Ye, L. Zhang, L. Wang, and X. Li. An Empirical Study on Detecting and Fixing Buffer Overflow Bugs. In Proceedings of the 2016 IEEE International Conference on Software Testing, Verification and Validation, 2016, pp. 91–101.

[10]. CVE security vulnerability database. Security vulnerabilities, exploits, references and more. Режим доступа: https://www.cvedetails.com/index.php, дата обращения: 2018-04-08.

[11]. K. Kratkiewicz and R. Lippmann. A taxonomy of buffer overflows for evaluating static and dynamic software testing tools. In Proceedings of Workshop on Software Security Assurance Tools, Techniques, and Metrics, vol. 500, 2006, pp. 44-51.

[12]. A. Borodin and A. Belevantcev. A static analysis tool Svace as a collection of analyzers with various complexity levels. Trudy ISP RAN /Proc. ISP RAS, vol. 27, issue 6, 2015, pp. 111–134. DOI: 10.15514/ISPRAS-2015-27(6)-8.

[13]. I.A. Dudina and A.A. Belevantsev. Using static symbolic execution to detect buffer overflows. Programming and Computer Software, vol. 43, no. 5, 2017, pp. 277–288. DOI: 10.1134/S0361768817050024.

[14]. Y. Zheng, X. Zhang, and V. Ganesh. Z3-str: A z3-based string solver for web application analysis. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, 2013, pp. 114–124.

Extracting architectural information from source code of ARINC 653-compatible application software using CEGAR-based approach

S.L. Lesovoy <lesovoy@ispras.ru>
Ivannikov Institute for System Programming of RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. It may be useful to analyze and reuse some components of legacy systems during development of new systems. By using a model-based approach it is possible to build an architecture model from the existing source code of the legacy system. The purpose of using architecture models is to analyze the system's static and dynamic features during the development process. These features may include real-time performance, resources consumption, reliability etc. The architecture models can be used as for system analysis as well as for reusing some components of the legacy system in the new design. In many cases it will allow to avoid creation of a new system from scratch. For creation of the architectural models various modeling languages can be used. In the present work Architecture Analysis & Design Language (AADL) is used. The paper describes an algorithm of extracting architectural information from source code of ARINC 653-compatible application software. ARINC 653 specification defines the requirements for software components of Integrated Modular Avionics (IMA) systems. To access the various services of ARINC 653 based OS an application software uses function calls defined in the APplication/Executive (APEX) interface. Architectural information in source code of application software compliant with ARINC 653 specification includes different objects and their attributes such as processes in each partition, objects for interpartition and intrapartition communications, as well as global variables. To collect the architectural information, it is necessary to extract all APEX calls from source code of application software. The extracted architectural information can be further used for creation the architecture models of the system. For source code analysis an approach based on Counterexample-guided abstraction refinement (CEGAR) algorithm is used. CEGAR algorithm explores possible execution paths of the program using its representation in the form of Abstract Reachability Graph (ARG). In a classical CEGAR algorithm a path in a program to be explored is called a counterexample and it means a path to the error state. In CPAchecker tool the basic predicate-based CEGAR algorithm has been extended for explicit-value analysis. In this paper the extended for explicit-value analysis CEGAR algorithm is applied for the task of extracting architecture information from source code. The main contribution of this paper is the application the ideas of counterexample and path feasibility check for the task of extracting the architectural information from source code.

Keywords: architectural information, architecture models, ARINC 653, IMA, CEGAR

DOI: 10.15514/ISPRAS-2018-30(3)-3

For citation: Lesovoy S.L. Extracting architectural information from source code of ARINC 653-compatible application software using CEGAR-based approach. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 31-46. DOI: 10.15514/ISPRAS-2018-30(3)-3

1. Introduction

The purpose of using architecture models is to analyze the system's static and dynamic features during the development process. These features may include real-time performance, resources consumption, reliability etc. This aspect is extremely important while developing complex systems that include both software and hardware components produced by the different suppliers. Using model-based approach at the early stages of the development of the project will help to avoid a waste of time and money for correction of system defects when the system is created. For creation of the architectural models various modeling languages can be used. The most popular ones used for architecture modelling are SysML[1] and AADL[2,3].

The model-based development process includes two major project steps. At the first step, the system model is being created. There are different levels for representation of the system model. The primary focus of this paper is the architectural models. On the second step of the project the system model will be used as input for detailed design and system implementation. This step may also include the model transformations to some intermediate formats used in system design and implementation. In the ideal case, the system model can be transformed to the source code of the system.

It may be useful to analyze and reuse some components of legacy systems during development of new systems. By using a model-based approach it is possible to build an architecture model from the existing source code of the legacy system. This model can be used as for system analysis as well as for reusing some components of the legacy system in the new design. In many cases, it will allow to avoid creation of a new system from scratch.

A process of model creation for existing system is called a model-driven reverse engineering (MDRE). If the source code of a legacy system is available then it is possible to build a system model from its source code. This process contains two steps. The first step is source code analysis. The second step is model transformations to the target output format. This paper describes the first step – source code analysis for application software that is based on Integrated Modular Avionics (IMA) architecture and ARINC 653 specification. The goal of source code analysis is to extract architecture information that is necessary for creation of the architecture model of the system.

The rest of the paper is organized as follows. Section 2 provides an overview of IMA architecture and ARINC 653 specification. It also contains a simple example of source code to be used for further analysis. Section 3 describes the concept of

architectural information in source code, a general approach and a particular algorithm used for extracting architectural information from source code. Section 4 describes the results and outlines the future research and development tasks.

2 IMA

IMA architecture is widely used in avionics industry for implementation the safety critical applications. In IMA systems multiple avionics applications can share resources of a single hardware platform (core module) without any mutual influence. ARINC 653 [4] is a set of documents that define the requirements for software components of IMA systems. The key concept of ARINC 653 is a partition. ARINC 653 compatible Operating System (OS) provides a dedicated portion of memory and predefined time slot within a fixed schedule for each partition. It prevents any affect from software executing in one partition to software in other partitions.

ARINC 653 specification defines that IMA system may include the following software components: core software, application partitions and system partitions. Core software consists of OS and APplication/EXecutive (APEX) interface. The APEX interface defines a set of services provided by the OS for application software. In each application, partition can be allocated only one application. System partitions contain system software that can directly interact with the OS without using APEX interface.

Communications between applications allocated in different partitions is called the interpartition communication. The interpartition communication is only available via communication channels. To access a communication channel the application can use the ports created inside a partition. ARINC 653 supports two port types: sampling ports and queuing ports.

An application software compliant with ARINC 653 specification has a typical structure. Such a software can be located in a single partition or in multiple partitions. To access the various services of ARINC 653 based OS an application software uses function calls defined in the APEX interface. For each partition several processes can be created. One process is responsible for partition initialization. This process creates other processes and various objects. Finally, when the initialization of partition has been finished this process sets the partition to NORMAL state using SET_PARTITION_MODE function call. Since this moment a scheduling for all processes created inside a partition is started. It is important to note that after the initialization of partition has been finished there is no way to create any new processes and objects.

An ARINC 653 process is quite similar to a POSIX thread. To create a process, it is necessary to create a structure that contains the process's attributes and pass it to CREATE_PROCESS function. ENTRY_POINT is an attribute of the process that contains the address of the function that will be called when the process is started. This function implements the application logic of the process and its communication procedures with other processes.

Communications between processes within a single partition is called intrapartition communication. Buffers and blackboards are used for communication between processes inside a partition. Semaphores, events and mutexes are used for process synchronization. Any objects for communication and synchronization can be created using function calls defined in the APEX interface. The processes located inside the same partition can also communicate via global variables.

At the end of this section, a simple example of application software will be demonstrated and explained. The source code fragment of the application software compliant with ARINC 653 specification is shown in Fig.1. Source code fragment in Fig.1 includes three functions: Run_10_Hz, Run_Monitor and main. In function main two processes, one event and three sampling ports are created.

```
static void Run_10_Hz(void) {
    …
    while (1) {
        SET_EVENT ( wakeup, ret );
        READ_SAMPLING_MESSAGE(port_raw_data,
             (MESSAGE_ADDR_TYPE)&sensor_data,
             &len, &validity, &ret);
        // Some operations with data …
        WRITE_SAMPLING_MESSAGE(port_data_out,
             (MESSAGE_ADDR_TYPE)&output_data,
             &len2, &ret);
        PERIODIC_WAIT(&ret_pause); }
}
static void Run_Monitor(void) {
    while (1) {
        WAIT_EVENT ( wakeup, TimeOut, ret );
        RESET_EVENT ( wakeup, ret );
        // Some operations with data …
        WRITE_SAMPLING_MESSAGE( port_status,
             (MESSAGE_ADDR_TYPE)&status_data,
            &len, &ret ); }
}
void main(void) {
    PROCESS_ATTRIBUTE_TYPE Proc_10_Hz_Attributes;
    Proc_10_Hz_Attributes.ENTRY_POINT = Run_10_Hz;
    Proc_10_Hz_Attributes.PERIOD = 100000000LL;
    strncpy(Proc_10_Hz_Attributes.NAME, "Proc_10_Hz",
        sizeof(PROCESS_NAME_TYPE));
    CREATE_PROCESS( &Proc_10_Hz_Attributes, &pid_p0,
        &ret );
    START( pid_p0, &ret );
```

```
    PROCESS_ATTRIBUTE_TYPE Proc_Monitor_Attributes;
    Proc_Monitor_Attributes.ENTRY_POINT = Run_Monitor;
    Proc_Monitor_Attributes.PERIOD =
        INFINITE_TIME_VALUE;
    strncpy(Proc_Monitor_Attributes.NAME, "Proc_Monitor",
        sizeof(PROCESS_NAME_TYPE));
    CREATE_PROCESS( &Proc_Monitor_Attributes, &pid_p1,
        &ret );
    START( pid_p1, &ret );

    EVENT_NAME_TYPE EventName;
    strncpy( EventName, "Wakeup", …);
    CREATE_EVENT ( EventName, wakeup, ret );
    …
    CREATE_SAMPLING_PORT( "RAW_DATA",
        port_size, DESTINATION, period, &port_raw_data, …);
    CREATE_SAMPLING_PORT( "DATA_OUT",
        port_size, SOURCE, period, &port_data_out, …);
    CREATE_SAMPLING_PORT( "STATUS", port_size,
        SOURCE, period, &port_status …);
    SET_PARTITION_MODE ( NORMAL, &ReturnCode );
    return 0;
}
```

Fig. 1. Source code fragment with APEX calls.

For process creation, APEX call CREATE_PROCESS is used. The first argument of CREATE_PROCESS has a type PROCESS_ATTRIBUTE_TYPE. It is a structure that contains attributes for the created process. The ENTRY_POINT attribute is equal to Run_10_Hz for the first process and is equal to Run_Monitor for the second one. Run_10_Hz and Run_Monitor are the function's names that are called when the processes are started.

Below in the main function, APEX call CREATE_EVENT is used to create an event object. An event object has a name Wakeup. Then APEX calls CREATE_SAMPLING_PORT are used to create three sampling ports. These ports have the following names: RAW_DATA, DATA_OUT and STATUS. In the end of the main function APEX call SET_PARTITION_MODE is used to set the partition to the NORMAL state. After that, OS will invoke functions Run_10_Hz and Run_Monitor.

A function Run_10_Hz is called periodically with period 10 milliseconds. This value for period was set in PERIOD attribute during the creation of the first process. Each time when the function Run_10_Hz is called, it activates the event Wakeup, reads a

message from sampling port RAW_DATA, performs some operations with data and writes a message to sampling port DATA_OUT.

Function Run_Monitor belongs to the second process that is an aperiodic. This function waits for event Wakeup, resets it, performs some operations with data and writes a message to sampling port STATUS.

3 Source code analysis

3.1 Architectural information in source code

The main goal of source code analysis in the paper is to extract the architectural information from it. Architectural information in source code of application software compliant with ARINC 653 specification includes the processes in each partition and their attributes, all objects created for interpartition and intrapartition communications and their attributes. It also includes the ways of communications and synchronizations between processes located inside the same partition or in different partitions. If the global variables are used for communication between processes inside partition then these variables also should be considered as architectural information.

The source code fragment in Fig.1 contains the following architectural information: two processes, one event and three sampling ports. Attributes of each process and each object (event, port) are also important architectural information. For synchronization between two processes the event object is used. In the first process APEX call SET_EVENT is used to activate an event. The second process uses APEX call WAIT_EVENT for receiving this event. Sampling ports are used in both processes to communicate with external environment, i.e. with processes allocated in other partitions or with external devices.

The source code of real avionic application can contain hundreds of processes communicating with each other and with external environment via large number of the objects. Extracting such architectural information from source code can be time consuming task. This paper proposes a way to do it automatically. The next sections describe a general approach and a particular algorithm used for source code analysis.

3.2 General approach for source code analysis

For source code analysis, an approach based on Counterexample-guided abstraction refinement (CEGAR) algorithm is used. In CPAchecker tool [5] the basic predicate-based CEGAR algorithm has been extended for explicit-value analysis [7]. CPAchecker is a tool for configurable program analysis (CPA) [5,6] that combines the traditional program analyses and software model checking. In this paper the extended for explicit-value analysis CEGAR algorithm is applied for the task of extracting architecture information from source code. The algorithm is implemented in CPAchecker tool.

The algorithm presented in this paper uses a Control-Flow Automata (CFA) as intermediate representations of the program to be analyzed. CFA is a directed graph

containing nodes and edges. A node corresponds to a program location. An edge corresponds to a certain operation of the program, for example, an assignment statement, a conditional branch or a function call. During the analysis, the algorithm constructs an Abstract Reachability Graph (ARG) using a program CFA. ARG is also a directed graph but its nodes correspond to abstract states of the program. Each abstract state contains a program location, a data state and a call stack. A data state is a mapping between program variables and their values. In data state some program variables may not have the values.

ARG represents possible execution paths of the program. It means that ARG can contain both feasible (real) program paths as well as the infeasible (spurious) paths. The program path is feasible if it can be executed at runtime otherwise it is infeasible. A path in ARG is a sequence of abstract states connected by edges. An abstract state is reachable if there is a feasible program path that contains this state.

3.3 Extracting APEX calls from source code

Before starting the algorithm description, it is necessary to explain some important concepts used by the algorithm. The algorithm constructs the ARG by sequentially adding the new abstract states to it. For the current state the algorithm gets the list of all its successors and adds each of them to ARG. There is an edge between the current state and each its successor.

Target states.

Some edges may correspond to a function call in source code. If this function is defined in APEX interface, the algorithm will need to collect additional information about this function call.

An abstract state in ARG which immediately follows such a function call is called the target state. Any target state has an incoming edge with APEX call. For each target state there is a path in ARG from the initial state to it. The algorithm performs a feasibility check for these paths.

Precision.

Explicit-value analysis tracks values for the program variables. In many cases it is enough to track only a part of program variables that are important for a particular analysis. A set of program variables that are being tracked for the current abstract state is called a precision. Different abstract states may have different precisions. The empty precision means that no variables are being tracked. The full precision means that all variables are being tracked. As described in [6] the value analysis algorithm implemented in CPAchecker can change a precision during the analysis depending on some conditions. It is called a precision adjustment.

An edge in ARG can correspond to a program operation that changes a value of a program variable. For example, an assignment operation changes the value of the left-hand operand, for a function call the values of arguments are assigned to function's parameters, etc. When the algorithm handles an edge between the current state (predecessor) and next state (successor) it uses a precision of the predecessor. If

precision of the predecessor contains the current variable, then the algorithm evaluates and stores its new value in abstract state. The algorithm of analysis can use the values of variables stored in abstract states for different purposes.

Fig. 2 presents a pseudocode of the main algorithm for extracting the architectural information from source code. This algorithm implements a classical CEGAR cycle extended for explicit value analysis [7] and is applied for the task of extracting architectural information from source code.

CFA of the program is used as an input data for the algorithm. The algorithm uses two variables to store the abstract states: “reached” and “waitlist”. A variable “reached” contains the set of abstract states that have been explored already. A variable “waitlist” contains the set of abstract states that have to be explored on the next steps of the algorithm.

At the beginning, the algorithm takes the initial state from CFA and put it to “waitlist”. After that, the external loop of the algorithm begins. The algorithm takes and removes the current state from waitlist. Further the algorithm gets all reachable successors for the current state using function “getAbstractSuccessors”. A pseudocode for the function “getAbstractSuccessors” is shown on Fig.3. The first operation of the function gets all successors (CFA nodes) of the current state. Then the function consecutively handles the edges (function “handleEdge”) between the current state and each its successor. The function “handleEdge” takes two parameters. The first parameter is an edge to be explored. The second parameter is a precision. The precision is taken from the edge predecessor. Depending on the operation in source code that the edge corresponds to, the function “handleEdge” performs the following actions:

- For an assignment operation, the algorithm evaluates a new value for this variable. The new value for a variable will be stored in abstract state if this variable is contained in the precision.
- For a function call, the function’s arguments are assigned to function’s parameters.
- For a conditional branch, a logical value for a condition is evaluated. If the logical value of a conditional branch is equal to FALSE then the function “handleEdge” return FALSE. It means that this successor is not reachable. In all other cases the function returns TRUE and the current successor is added to the list of reachable successors. So, function “getAbstractSuccessors” returns for the current state a list of all its reachable successors.

```
FUNCTION main
INPUT
    CFA of the program;
OUTPUT
```

```
    Architectural information
VARIABLES
    reached – a set of states that have been reached;
    waitlist – a set of states to be explored;
BEGIN
    initState = getInitialState(CFA);
    addStateToWaitlist(initState);
    // Traverse through all CFA nodes.
    LOOP WHILE waitlist ≠ 0 // External loop.
        curState = getAndRemoveStateFromWaitlist();
        // Get all reachable successors of the current state.
        successors = getAbstractSuccessors(curState);
        // Traverse through all reachable successors.
        FOR EACH nextState IN successors // Internal loop.
            IF isTargetState(nextState)
                path = getPathToState(nextState);
                IF isPathFeasible(path) = FALSE
                    // Refine the path.
                    performRefinementForPath(path,
                        reached, waitlist);
                    BREAK // Go to external loop.
                END IF
            END IF
            merge(nextState, reached);
            update(reached);
            addStateToWaitlist(nextState);
        END FOR EACH
    END LOOP
END
```

Fig. 2. The main algorithm for extracting the architectural information from source code

Further in internal loop the main algorithm traverses through all reachable successors for the current state. At this part of algorithm, a successor is called as a “nextState”. The algorithm checks whether a nextState is a target state. If it is a target state, the algorithm calculates a path in ARG from the initial state to the current target state and checks its feasibility using function “isPathFeasible”. In a classical CEGAR algorithm a path in a program to be explored is called a counterexample and it means a path to the error state. In the current algorithm it is just a path to the target state we need to explore.

The algorithm of function “isPathFeasible” is shown in Fig. 4. To check the path

feasibility the algorithm consecutively passes through all edges of the path, starting from the initial state. The algorithm analyses the operations for each edge. To track all program variables, for each state on the path the full precision is set, i.e. the algorithm performs the feasibility check for a path with the full precision.

```
FUNCTION getAbstractSuccessors
INPUT
    curState // Current state.
RETURN
    reachableSuccessors // All reachable successors.
BEGIN
    // Get all successors of the current state.
    allCFASuccessors = getAllSuccessors(curState);
    FOR EACH successor IN allCFASuccessors
        edge = getEdge(curState, successor);
        precision = getPrecisionForState(curState);
        IF handleEdge(edge, precision) = TRUE
            // Add  successor to reachableSuccessors.
            addToSet(reachableSuccessors, successor);
        END IF
    END FOR EACH
    RETURN reachableSuccessors;
END
```

Fig. 3. The algorithm of function getAbstractSuccessors

Each edge on the path is handled with the function “handleEdge” that was already described above. For a conditional branch the function “handleEdge” may return FALSE if logical condition is not satisfied. The path is not feasible if for any edge on the path the logical condition is not satisfied. In this case the function “isPathFeasible” returns FALSE. In all other cases the path is feasible. If the path is feasible then at the last state of the path the values for all program variables assigned on this path are known. The last edge and the last state of the path is passed to a function “handleApexCall”.

```
FUNCTION isPathFeasible
INPUT
    path
RETURN
    TRUE – path is feasible;
```

```
    FALSE – path is not feasible;
BEGIN
    // Traverse through all edges.
    FOR EACH edge IN path
        precision = FULL;
        IF handleEdge(edge, precision) = FALSE
            RETURN FALSE
        END IF
        IF isLastEdge(edge, path) = TRUE
            lastState = getSuccessor(edge);
            handleApexCall(edge, lastState);
        END IF
    END FOR EACH
    RETURN TRUE
END
```

Fig. 4. The algorithm of function isPathFeasible

The last edge contains the information about the APEX call. The last state contains values for all program variables on the path. The function "handleApexCall" extracts all architectural information including the function name for the last APEX call, values for its argument and call stack. It is important to note that the algorithm extracts architecture information only from the APEX calls that belong to a feasible paths. The algorithm collects the architectural information for each APEX call and uses it as output data. The format of the output data will be described in the next section.

If the algorithm has detected that a path is infeasible, then it will refine this path. During refinement procedure the precision for some abstract states of the path are changed by adding variables for tracking. The refinement procedure is described in detail in [7]. Finally, the algorithm will update the ARG in such a way that will eliminate the infeasible path or its part for the further analysis.

At the end of the internal loop the algorithm tries to merge the nextState with already reached states, updates reached states and adds the last explored state (nextState) to "waitlist". These steps are described in details in [7] (see section "Reachability Algorithm for CPA").

The described above steps of internal loop are being repeated for each reachable successor of the current state.

Then the algorithm leaves the internal loop and continues its execution by taking the first step on the main loop. It takes the next state from "waitlist" variable and repeats all steps already described above. The algorithm terminates when all ARG abstract states have been processed.

3.4 Output format

The algorithm keeps the collected architectural information in the internal format. For further processing the architectural information has to be transformed to the external representation. The export format depends on the tool that is used for creation the architecture models. The architectural information can also be exported to human-readable format. In Fig. 5 the architectural information extracted by the algorithm from the source code fragment in Fig. 1 is presented in a human-readable format. The presented architectural information is divided onto sections. The first section contains information about ARINC653 processes. There are two processes with names Proc_10_Hz and Proc_Monitor. Below the process name there are the list of its attributes. On the Fig3 there are only three attributes are presented: PROCESS_ID, ENTRY_POINT and PERIOD. PROCESS_ID is a serial number of the process inside a partition. ENTRY_POINT is a name of the function that is being called when the process is started. PERIOD shows the period's duration in milliseconds. INFINITE_TIME_VALUE in source code corresponds to aperiodic process. The next sections contain the information about other ARINC653 objects created in the source code.

ARINC653_SAMPLING_PORTS section shows three sampling ports and its attributes.

ARINC653_SAMPLING_MESSAGES section shows what processes are using sampling ports for sending (WRITE subsection) and for receiving (READ subsection) messages. For example the port DATA_OUT is used by the first process (function Run_10_Hz) for sending messages.

ARINC653_EVENTS contains information about the events that have been created and used in the source code.

ARINC653_EVENTS section has three subsections: SET_EVENT, WAIT_EVENT and RESET_EVENTS. The name of the subsection corresponds to the APEX call. For example, a subsection SET_EVENT corresponds to APEX call SET_EVENT that activate an event.

```
==ARINC653_PROCESSES==
Proc_10_Hz
  PROCESS_ID: 0
  ENTRY_POINT: Run_10_Hz(0)
  PERIOD = 100 ms
  …
Proc_Monitor
  PROCESS_ID: 1
  ENTRY_POINT: Run_Monitor(1)
  APERIODIC
  …
==ARINC653_SAMPLING_PORTS==
```

```
1) RAW_DATA
  MAX_MESSAGE_SIZE = 128
  PORT_DIRECTION = DESTINATION
  REFRESH_PERIOD = 1000
…
2) DATA_OUT
…
3) STATUS
…
==ARINC653_SAMPLING_MESSAGES==
=WRITE=
1) PORT_NAME=DATA_OUT;
    ENTRY_POINT=Run_10_Hz(0);
2) PORT_NAME=STATUS;
    ENTRY_POINT=Run_Monitor(1);
=READ=
1) PORT_NAME=RAW_DATA;
    ENTRY_POINT=Run_10_Hz(0);

==ARINC653_EVENTS==
=SET_EVENT=
1) EVENT_NAME=Wakeup;
    ENTRY_POINT=Run_10_Hz(0)
=WAIT_EVENT=
1) EVENT_NAME=Wakeup;
    ENTRY_POINT=Run_Monitor(1)
=RESET_EVENTS=
...
```

Fig. 5. The architectural information in human-readable format.

In the analyzed source call there is only one such a call for event with a name Wakeup. The ENTRY_POINT string contains a name of the ENTRY_POINT function where this call was made. In the real code the ENTRY_POINT function is determined using a call stack information. The serial number of the process is shown in parentheses. In the Fig.3 we can see that event Wakeup was set in function Run_10_Hz that belongs to the process with PROCESS_ID equal to 0 (Proc_10_Hz). From the section WAIT_EVENT, we can understand that the function Run_Monitor waits for the event Wakeup using APEX call WAIT_EVENT. The function Run_Monitor belongs to process Proc_Monitor. So, we can see that the event Wakeup is used by two processes for synchronization.

The representation of architectural information in the human-readable format is presented only for explaining the content of such information and is useful mainly for debug purposes. As it was mentioned above for further processing the architectural

information should be transformed to the format that is supported by the external tools.

4 Results and conclusions

The algorithm presented in the paper allows extracting architectural information from source code of ARINC 653-compatible application software. The main contribution of this paper is the application the ideas of counterexample and path feasibility check for the task of extracting the architectural information from source code. In the presented algorithm the task of extracting architectural information from source code has been solved by transforming it into the task of path feasibility check.

The work of the algorithm is demonstrated on the simple example. By this moment the algorithm has been tested on the several software applications that are compatible with ARINC 653 specification. These applications contained up to 50 ARINC 653 process and up to 30 objects for communications.

The next task to be done is to extend the algorithm for extracting from source code the global variables that are used for communication between processes inside partition. It is also necessary to implement the algorithm of transformation of the architecture information to the architecture model.

References

[1]. OMG Systems Modeling Language (OMG SysML™) Version 1.5, 2017.
[2]. [Online]. Available: http://www.omg.org/spec/SysML/1.5/
[3]. SAE International standard AS5506C, Architecture Analysis & Design Language (AADL), 2017. [Online]. Available: http://standards.sae.org/as5506c/
[4]. Feiler P., Gluch D. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley, 2012.
[5]. ARINC Specification 653P1-4. Avionics Application Software Standard Interface Part 1 – Required Services. Published by SAE-ITC, Maryland, USA. August 21, 2015.
[6]. [Online]. Available: https://cpachecker.sosy-lab.org/
[7]. D. Beyer, T. A. Henzinger and G. Theoduloz. Program Analysis with Dynamic Precision Adjustment. In Proc, of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 29-38.
[8]. Beyer D., Löwe S. Explicit-State Software Model Checking Based on CEGAR and Interpolation. Lecture Notes in Computer Science, vol. 7793, pp 146-162.

Извлечение архитектурной информации из исходного кода ARINC 653 совместимых приложений с использованием алгоритма CEGAR

С.Л. Лесовой <lesovoy@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Модельно-ориентированный подход к разработке позволяет построить архитектурную модель существующей системы по ее исходному коду. Построенная архитектурная модель существующей системы позволяет проанализировать ее различные статические и динамические характеристики, включая производительность, требуемые аппаратные ресурсы, надежность и другие. Архитектурные модели могут использоваться как для анализа, так и для повторного использования некоторых компонентов существующей системы в новом проекте. Во многих случаях такой подход позволяет избежать построения новой системы с нуля. Для создания архитектурных моделей могут использоваться различные языки моделирования. В данной работе используется язык анализа и проектирования архитектуры (AADL). Данная статья описывает алгоритм извлечения архитектурной информации из исходного кода ARINC 653 совместимых программных приложений. Спецификация ARINC 653 определяет требования к программным компонентам для систем интегрированной модульной авионики (ИМА). Для доступа к различным сервисам операционной системы программные приложения используют прикладной исполняемый интерфейс. Архитектурная информация в исходном коде программных приложений совместимых с требованиями спецификации ARINC 653 включает процессы в каждом разделе, объекты для взаимодействия между процессами внутри и за пределами раздела, а также глобальные переменные. Для анализа исходного кода и получения архитектурной информации необходимо проанализировать все программные вызовы прикладного исполняемого интерфейса. Извлеченная архитектурная информация далее используется для построения архитектурных моделей системы. Для анализа исходного кода используется подход на основе алгоритма CEGAR (уточнение абстракции с помощью контрпримера), широко используемого при верификации программного обеспечения. Алгоритм CEGAR анализирует возможные пути исполнения программы, используя представление программы в виде абстрактного графа достижимости. В классическом алгоритме CEGAR исследуемый путь программы называется контрпримером и означает путь от начала программы до некоторого ошибочного состояния. Для подтверждения наличия ошибки в коде программы алгоритм CEGAR выполняет проверку достижимости для исследуемого пути. В программном инструменте CPAchecker базовый основанный на предикатах алгоритм CEGAR расширен для анализа явных значений переменных. В данной статье расширенный для анализа явных значений переменных алгоритм CEGAR используется для задачи извлечения архитектурной информации из исходного кода приложений. Основной вклад данной статьи заключается в применении идей контрпримера и проверки достижимости пути к задаче извлечения архитектурной информации из исходного кода приложений.

Ключевые слова: архитектурная информация; архитектурные модели; ARINC 653; интегрированная модульная авионика (ИМА); алгоритм CEGAR

DOI: 10.15514/ISPRAS-2018-30(3)-3

Для цитирования: Лесовой С.Л. Извлечение архитектурной информации из исходного кода ARINC 653 совместимых приложений с использованием алгоритма CEGAR. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 31-46 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-3

Список литературы

[1]. OMG Systems Modeling Language (OMG SysML™) Version 1.5, 2017.
[2]. [Online]. Режим доступа: http://www.omg.org/spec/SysML/1.5/
[3]. SAE International standard AS5506C, Architecture Analysis & Design Language (AADL), 2017. [Online]. Режим доступа: http://standards.sae.org/as5506c/
[4]. Feiler P., Gluch D. Model-Based Engineering with AADL: An Introduction to the SAE Architecture Analysis & Design Language. Addison-Wesley, 2012.
[5]. ARINC Specification 653P1-4. Avionics Application Software Standard Interface Part 1 – Required Services. Published by SAE-ITC, Maryland, USA. August 21, 2015.
[6]. [Online]. Режим доступа: https://cpachecker.sosy-lab.org/
[7]. D. Beyer, T. A. Henzinger and G. Theoduloz. Program Analysis with Dynamic Precision Adjustment. In Proc, of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008, pp. 29-38.
[8]. Beyer D., Löwe S. Explicit-State Software Model Checking Based on CEGAR and Interpolation. Lecture Notes in Computer Science, vol. 7793, pp 146-162.

Вопросы индустриального применения синхронизационных контрактов при динамическом поиске гонок в Java-программах

В.Ю. Трифанов <vitaly.trifanov@gmail.com>,
Санкт-Петербургский государственный университет,
198504, Россия, Санкт-Петербург, Университетский пр., д. 28

Аннотация. Состояния гонки (data race) возникает в многопоточной программе при одновременном обращении нескольких потоков к разделяемым данным. Существует два основных подхода к обнаружению гонок – статический анализ программы (без её запуска) и динамическое обнаружение гонок в процессе работы программы. Ранее авторами был предложен точный высокопроизводительный динамический подход к обнаружению гонок на основании специальным образом составленных синхронизационных контрактов – частичных спецификаций поведения классов и наборов методов целевого приложения в многопоточной среде. Данная статья рассматривает вопрос индустриального применения концепции синхронизационных контрактов на крупных нагруженных многопоточных приложениях. Предложены метод обработки контрактов и архитектура соответствующего модуля динамического детектора jDRD, выявлены основные проблемные места и потенциальные точки падения производительности, разработано техническое решение, лишённое подобных проблем.

Ключевые слова: состояние гонки; многопоточность; динамический анализ; автоматическое обнаружение ошибок.

DOI: 10.15514/ISPRAS-2018-30(3)-4

Для цитирования: Трифанов В.Ю. Вопросы индустриального применения синхронизационных контрактов при динамическом поиске гонок в Java-программах. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 47-62. DOI: 10.15514/ISPRAS-2018-30(3)-4

1. Введение

Многопроцессорные и многоядерные системы начали своё развитие в конце XX века и на текущий момент фактически вытеснили однопроцессорные вычислительные машины. Основным преимуществом подобных систем является возможность одновременного выполнения различных

последовательностей инструкций параллельно. Наиболее популярной программной архитектурой таких систем является модель разделяемой памяти, в которой несколько потоков управления обмениваются данными через общую память. Такая архитектура лежит в основе таких широко распространённых в индустрии языков как C++ и Java [1].

Организация корректного взаимодействия нескольких потоков является одной из самых сложных задач в программировании, здесь возможны серьёзные ошибки, такие как взаимные блокировки (deadlocks), голодание (thread starvation) и гонки (data races).

Гонка (или состояние гонки) возникает в многопоточной программе, когда несколько потоков одновременно обращаются к разделяемым данным, и хотя бы одно из этих обращений является записью данных [2]. Основная опасность гонок заключается в том, что они приводят к повреждению глобальных структур данных, но программа при этом зачастую продолжает работу, что приводит к сбоям и непредвиденным результатам. Ущерб от гонок может приводить к огромным финансовым потерям и к человеческим жертвам [3, 4]. Возникновение гонки зависит от чередования операций в потоках. При внешнем управлении потоками (а именно так устроено подавляющее большинство систем) переключение между ними происходит непредсказуемо, что затрудняет возможность воспроизведения гонок на стадии тестирования. Таким образом, задача автоматического обнаружения гонок актуальна, важна и находится в поле внимания исследователей на протяжении нескольких десятилетий.

Принято выделять два основных подхода к автоматическому обнаружению гонок. Первый – это статический подход, который предполагает анализ исходного кода программы без её запуска. Например, возможно построение графа выполнения программы (control flow graph, CFG) и расчёт множества удерживаемых потоками блокировок. Такой алгоритм называется lockset [5, 6] и обладает существенным недостатком – с его помощью можно отслеживать только операции синхронизации, основанные на блокировках. В современном программировании это становится все менее актуально, поскольку разработаны неблокирующие операции синхронизации и типовые структуры данных, такие как очередь и хеш-таблица [7]. В простых случаях можно успешно доказать корректность программ без блокировок с помощью тяжеловесных методов – например, алгоритма Деккера или Петерсона. Анализ использования объектов и контекста [8-11] также позволяет корректно обрабатывать некоторые другие способы синхронизации. Также возможен анализ с помощью проверки моделей [12-13] или использования дополнительных типов [14] и их вывода с помощью аннотаций [15-16].

Второй подход – динамический, в рамках которого анализ программы осуществляется во время её выполнения [17-24]. Он анализирует лишь текущий путь выполнения программы, но в нём может обрабатывать любые операции синхронизации и обладать стопроцентной точностью, без ложных

срабатываний 1-го и 2-го рода. Однако на практике его точность сильно ограничена производительностью, поскольку обработка всех операций синхронизации и обращений к разделяемым данным требует в сотни раз больше ресурсов, чем выполнение самой анализируемой программы. Следует отметить, что последние разработки в сфере динамического поиска гонок сводятся к сокращению области анализа с помощью различных техник, в том или ином виде жертвующих точностью анализа [25-26].

Ранее авторами был предложен подход *синхронизационных контрактов*, который позволяет существенно повысить производительность динамического обнаружения гонок без потери точности [27-29]. Этот подход показал хорошие практические результаты, но в процессе его индустриального применения обнаружился ряд трудностей. Во-первых, созданные разработчиками контракты могут содержать неточности и ошибки, а во-вторых, контракты нужно эффективно применять в процессе анализа кода, чтобы достичь экономии накладных расходов на сбор информации и проверку гонок во время работы приложения.

В данной статье предложен подход к анализу, верификации и динамическому применению контрактов, основанный на построении дерева контрактов (аналог графа потока управления), и продемонстрирована его целесообразность и практическая полезность.

2. Синхронизационные контракты и детектор jDRD

Основным точным алгоритмом динамического обнаружения гонок в программах, основанных на модели разделяемой памяти, является алгоритм happens-before [18, 30], являющийся, фактически, поиском гонок «по определению». Алгоритм, анализируя операции в программе, выделяется два подмножества: (i) операции с данными внутри потоков исполнения и (ii) операции синхронизации, передающие изменения между потоками и таким образом синхронизирующие данные различных потоков.

Рассмотрим работу алгоритма на примере языка Java, у которого, наряду с C/C++, модель памяти максимально проработана [1]. На множестве всех операций синхронизации существует отношение полного порядка synchronized-with. Все операции внутри каждого конкретного потока также упорядочены – это порядок по времени. Объединение и последующее транзитивное замыкание этих двух отношений даёт отношение частичного порядка, называемое happens-before. Согласно определению гонки, две операции с одними и теми же данными находятся в состоянии гонки, если они не упорядочены с помощью отношения happens-before [1].

Традиционно выполнение отношения happens-before отслеживается с помощью векторных часов Лампорта [31]. Такой подход точен, но обладает очень большими накладными расходами – скорость работы программы замедляется в 10-200 раз [32]. Для повышения производительности ранее

авторами был предложен подход синхронизационных контрактов, в основе которого лежит два наблюдения.

1. Индустриальные приложения используют сторонние библиотеки и подсистемы, которые в совокупности могут превышать размер кода самого приложения в несколько десятков раз. Как правило, взаимодействие со сторонними библиотеками осуществляется через хорошо документированные интерфейсы. В частности, обычно хорошо документировано поведение методов, классов и интерфейсов в многопоточной среде.
2. При поиске гонок обычно стоит задача обнаружения ошибок в собственном коде приложения, поскольку выбор библиотек обычно осуществляется экспертами на основании больших объёмов данных об их предыдущем использовании. Это позволяет сделать вывод о существенно меньшей надёжности самого разрабатываемого приложения, чем используемых им библиотек и подсистем, и сфокусироваться на анализе собственного кода приложения.

На основе этих наблюдений был разработан метод синхронизационных контрактов, а также динамический детектор jDRD [27-29, 33].

Основная идея метода заключается в том, чтобы разделить весь программный код целевого приложения на две части – подлежащую анализу (обычно это собственный код приложения) и не подлежащую анализу, то есть считающуюся надёжной (обычно это сторонние библиотеки, подсистемы и модули). Далее необходимо определить интерфейсы и методы, посредством которых анализируемая часть взаимодействует с неанализируемой частью, и описать их поведение в многопоточной среде. Иными словами, нужно выделить используемые в приложении интерфейсы других компонент, проанализировать и классифицировать их классы и методы там в тех случаях, где про них что-то известно. В [33] предлагается следующая классификация:

1. вызов пары методов (или несколько пар методов одного класса) обеспечивают передачу отношения happens-before согласно документации;
2. метод или класс потокобезопасен, но не вовлечён в передачу отношения happens-before;
3. метод объекта не предназначен для вызова в многопоточной среде (требует внешней синхронизации), но может рассматриваться как немодифицирующий состояние объекта-владельца (то есть, может рассматриваться как операция чтения данных).

Единичное описание принадлежности метода (или всех методов класса) к какой-то категории и называется синхронизационным контрактом. Контракт для пары методов, обеспечивающих передачу отношения happens-before, называется happens-before контрактом. Точность и ограничения такого подхода описаны в работе [28], там же представлены основные виды

контрактов и продемонстрировано повышение производительности динамического анализатора, увеличивающееся с сокращением анализируемой области и ростом базы контрактов. В последующей статье [29] представлен язык описания контрактов, позволяющий разрабатывать синхронизационные контракты отдельных методов, пар методов или классов. Кроме того, язык содержит директивы, позволяющие включить контракты из других файлов. Это обеспечивает переиспользуемость контрактов – достаточно описать один раз контракты библиотеки и внести файл с ними в комплект поставки. На листинге 1 представлен пример контракта, указывающего наличие отношения happens-before между записью в потокобезопасную хеш-таблицу o по некоторому ключу p1 и последующими чтениями из неё по тому же ключу.

```
sync {
        key java.lang.Object=o, java.lang.Object=p1;
        send
java.lang.Objectjava.util.concurrent.ConcurrentMap.put(java.lang.Object,
java.lang.Object);
        receive
java.lang.Objectjava.util.concurrent.ConcurrentMap.get(java.lang.Object);
}
```

Листинг 1. Пример контракта
Listing 1. Contract example

В дальнейшем эти контракты используются на фазе динамического анализа: перед вызовом метода детектор определяет, есть ли контракт для этого метода, и если есть, то обрабатывает метод в соответствии с этим контрактом, что и даёт прирост производительности (в противном случае пришлось бы проводить анализ всего кода метода).

В остальном схема устройства jDRD достаточно стандартна. На фазе запуска анализируемого приложения к нему посредством стандартной технологии java-agent подключается модуль jDRD, который модифицирует загружаемые классы с помощью техники инструментирования байт-кода. В код классов вставляются инструкции для обработки операций синхронизации и обращений к разделяемым данным. Во время выполнения таких операций управление передаётся в jDRD, который динамически обсчитывает векторные часы и обнаруживает гонки.

3. Проверка корректности контрактов

На этапе индустриального внедрения описанной выше технологии возник ряд сложностей, требующих правильных архитектурных и технических решений. В основной массе они связаны с тем, что в общем случае контракты создаются на основе различных источников (конфигурационные файлы, аннотации в коде, документация и т.д.), и зачастую различными специалистами. Это неизбежно приводит к возможности различных ошибок – например, неполноте, противоречивости или некорректности совокупного набора

контрактов – и поэтому после объединения всех контрактов воедино необходимо верифицировать полученный набор. Выходными данными модуля обработки контрактов является изменённый код целевого приложения (инструментированный байт-код), учитывающий контракты при обработке операций в приложении. Таким образом, задача обработки контрактов состоит из двух следующих частей (процедур).

1. Загрузка, проверка корректности и размещение контрактов в памяти.
2. Применение контрактов и модификация кода целевого приложения.

Первая процедура осуществляется один раз при запуске приложения. Нет ограничений на производительность этой процедуры, её основной задачей является проверка корректности контрактов. Вторая процедура является критичной с точки зрения производительности и требует тщательной реализации: контракты будут применяться постоянно (сотни раз в секунду) на протяжении всего временя работы приложения.

Рассмотрим подробнее первую процедуру. Язык описания контрактов [29] предоставляет синтаксические директивы для спецификации контрактов, поэтому на стадии чтения этих контрактов необходимо провести соответствующие процедуры по их разбору и проверке корректности. Здесь возможны следующие ситуации.

1. Контракты могут противоречить друг другу; например, в одном контракте указано, что метод A синхронизирован с B, а во втором – что, наоборот, B синхронизирован с A; в подобных случая необходимо сигнализировать об ошибке.
2. Контракты классов могут быть неполными, т.е. описывать лишь часть публичных методов класса. В этом случае нужно выдать предупреждение.
3. Контракты могут дублироваться – в этом случае нужно убедиться в их семантической идентичности.

Для проверки непротиворечивости контрактов необходимо убедиться в отсутствии циклических контрактных зависимостей между ними. Для обнаружения таких зависимостей строится граф контрактов. Вершинами графа являются методы контрактов, а ребро между парой вершин A и B существует тогда и только тогда, когда A предшествует B. Согласно принципу подстановки Лисков [34] при наследовании потомок должен удовлетворять контракту предка. Следовательно, для каждого класса целевого приложения необходимо получить список его потомков и добавить соответствующие рёбра в граф контрактов. Наличие цикла в таком графе означает наличие цепочки противоречащих друг другу контрактов, а отсутствие циклов – непротиворечивость всего набора контрактов.

Для проверки контрактов классов на полноту на этапе загрузки контрактов анализируется каждый класс целевого приложения, для которого существует

контракт, и выясняется, все ли его публичные методы упомянуты в контракте. Если это не так, то выдаётся предупреждение.

Таким образом, процедура загрузки и проверки корректности контрактов имеет следующий вид.

1. Построить граф контрактов.
2. Для каждого контракта на пару методов (f, g) классов A и B добавить в граф контракты (f', g') для всех наследников классов A и B.
3. Выполнить проверку циклов в графе.
4. (Опционально). Для всех классов, упомянутых в контрактах, проверить наличие их публичных методов, не упомянутых в контрактах.

Данная процедура реализована в виде компоненты, предварительно обрабатывающей контракты до запуска целевого приложения. Она выдаёт дерево контактов и другую информацию, используемую в дальнейших динамических проверках.

4. Архитектура модуля применения контрактов

Рассмотрим процедуру применения контрактов с точки зрения её оптимальной реализации. Точнее, опишем программный модуль, который реализует эту процедуру.

Динамический детектор должен в режиме реального времени определять наличие контракта для определённого Java-метода. При этом скорость работы детектора не должна деградировать с ростом программы, увеличением числа контрактов или иерархии классов. Обратим внимание, что речь идёт о сотнях и тысячах операций в секунду, поскольку вызовы методов в Java-программах происходят регулярно. Соответственно, решение данной задачи требует нестандартных подходов как к организации структуры данных, управляющей контрактами, так и к техническим решениям по их проверке в режиме реального времени.

В результате построения дерева контрактов для каждого метода становится известен набор контрактов, в которые он вовлечён. Поэтому во время вызова этого метода в процессе работы программы детектор jDRD может эффективно получить данную информацию. Для внедрения соответствующих инструкций в код целевого приложения используется техника инструментирования байт-кода. Механизм её работы заключается в следующем: на фазе загрузки классов целевого приложения в оперативную память, после загрузки очередного класса управление передаётся компоненте Instrumentator – специальному Java-агенту, который реализован в рамках jDRD и может модифицировать байт-код данного класса (см. рис. 1). Instrumentator анализирует список методов класса и, если для метода существуют контракты, встраивает в качестве первой инструкции метода обработку этого контракта. Во время работы модифицированное приложение автоматически обработает

контракты всех методов (компонента RaceDetector). В практическом плане наличие контракта метода означает следующее.

1. Исходный код метода анализировать не нужно. Иными словами, во время выполнения тела метода детектор не должен проводить никаких операций и проверок. Для этого сразу после входа в тело метода детектор ставит специальный флаг в состояние «contract», а на выходе – сбрасывает. Перед тем, как выполнить очередную операцию, детектор проверяет состояние флага для текущего потока, и если его состояние равно «contract», то игнорирует операцию. Таким образом, флаг хранится для каждого потока отдельно. Управление флагами вынесено в отдельную компоненту (класс) FlagManager, основанный на стандартной Java-структуре ThreadLocal, которая предоставляет эффективный способ хранения локальных данных потока (см. рис. 1).
2. happens-before контракты подразумевают синхронизацию методов. Обработка этой информации реализована классическим способом – в виде векторных часов (компонента ClockStorage, см. рис. 1). Для хранения часов используется потокобезопасная хеш-таблица. Однако с учётом высокой нагрузки и большой частоты обращений традиционные потокобезопасные хеш-таблицы здесь не подходят. После ряда экспериментов была выбрана неблокирующая хеш-таблица [35]. Остаётся вопрос в выборе ключа для хранения часов. Ключ представляет собой отдельный объект, состоящий из полей, указанных в контракте. Поскольку набор полей от контракта к контракту может отличаться, классы объектов типа «ключ» генерируются автоматически, «на лету», посредством инструментирования байт-кода. На практике таких ключей нужно не более 20-30, поэтому на производительность динамического анализа это не оказывает существенного влияния. Отметим, что основная работа по анализу гонок производится в компоненте RaceDetector, которая работает только с векторными часами.

Единственным существенным недостатком описанного выше подхода является необходимость постоянного обращения к компоненте FlagManager. В частности, это регулярно происходит при работе самого jDRD, поскольку внутри него используются структуры данных, которые активно используют контракты. Одно такое обращение занимает порядка 1 мкс, но с учётом большого числа обращений (десятки-сотни тысяч в секунду) это существенно снижает производительность jDRD. Возможным решением может быть полный отказ от использования стандартных Java-классов во внутренних структурах детектора jDRD.

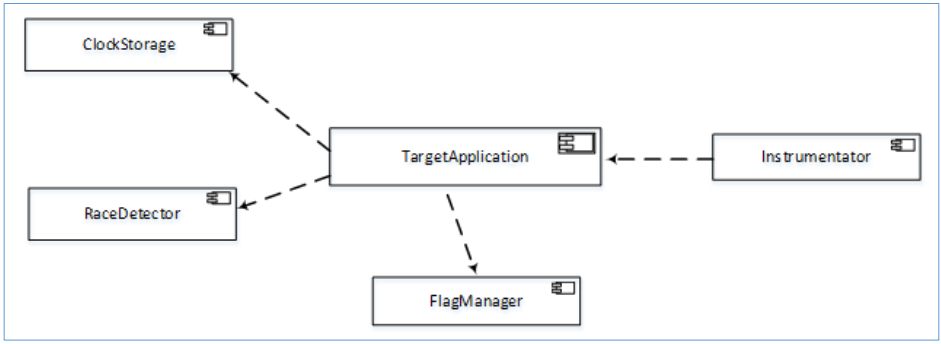


Рис.1. Архитектура модуля применения синхронизационных контрактов
Fig 1. Architecture of the contracts processing module.

5. Экспериментальное исследование

Предложенный подход был применён для анализа трёх приложений, имеющих разную специфику. В каждом содержалось 1000-2000 классов без учёта используемых библиотек, а также 10-30 потоков. Синхронизационные контракты для каждого приложения разрабатывались соответствующими командами программистов. Кроме того, использовался общий набор контрактов, разработанный ранее для стандартных средств синхронизации Java. Совокупный размер контрактов для каждого приложения не превышал долей процента от общего объёма кода приложения.

В табл. 1 представлены результаты экспериментов. В столбце «Количество контрактов» указано совокупное количество методов, для которых были составлены контракты. Столбец «количество ошибок» указывает число ошибок, обнаруженных при проверке корректности контрактов. Последний столбец содержит количество публичных методов, которые нуждались в составлении контракта, но были пропущены составителями.

Таблица 1. Результаты работы модуля обработки синхронизационных контрактов
Table 1. The contract processing module work results

Приложение	Кол-во контрактов	Кол-во ошибок	Пропущенные методы
A	80	0	75
B	135	3	95
C	120	1	80

Наборы контрактов в каждом случае описывали поведение порядка 500-1000 методов. Время работы модуля проверки занимает не более 10-20 секунд. Основная часть пропущенных методов приходилась на классы-структуры данных. Например, программист указывал метод get стандартной хеш-таблицы (HashMap) или списка (ArrayList) как немодифицирующий, но не указывал остальные немодифицирующие методы этого класса. Ещё в

нескольких случаях метод вспомогательного класса (utility class) указывался как потокобезопасный, в то время как таковым можно пометить весь класс.

Таким образом, внедрение модуля верификации контрактов оказалось полезным и позволило обнаружить несколько ошибок и ряд неточностей в составлении контрактов при достаточно малом времени работы.

Оценка производительности всего подхода синхронизационных контрактов на фазе динамического анализа является отдельной задачей, требующей, как минимум, разработки методики оценки. Дело в том, что, во-первых, динамический детектор jDRD совершает до ста тысяч операций в секунду[1]. Во-вторых, время, затрачиваемое на динамическую проверку контракта, может превосходить выигрыш от его применения.

Тем не менее, на основе проведенных экспериментов могут быть сделаны некоторые выводы по производительности. Во-первых, два из трёх приложений являются клиентскими, и пользователи показали повышение скорости работы приложений в сравнении с анализом при помощи старой версии jDRD. Во-вторых, общее число обрабатываемых операций синхронизации в секунду сократилось, поскольку количество добавившихся (обработка контрактов) на несколько порядков меньше количества устранённых (обработка операций синхронизации Java внутри контрактных методов). Как следствие, сократился и расход памяти на содержание векторных часов.

Полученные предварительные данные свидетельствуют о неухудшении производительности на анализируемых приложениях и о практической применимости предложенного подхода. Но требуются более детальные экспериментальные исследования.

6. Заключение

Динамический анализ является одним из основных подходов к автоматическому обнаружению гонок, но его практическая применимость ограничена вопросами производительности. Подход к снижению накладных расходов, основанный на описании синхронизационных контрактов сторонних компонент и замене их анализа применением этих контрактов, показал ранее высокую точность и практическую применимость. Однако как разбор контрактов, так и их применение на фазе динамического анализа связано с множеством сложностей. Возникают задачи как валидации и нормализации

[1]Отметим, что задача тестирования производительности на микро-уровне (так называемый микробенчмаркинг) невероятно сложна. Так, команда разработчиков Core Java работала над утилитой, позволяющей надёжно измерять производительность операций на уровне микро- и нано-секунд, около 5 лет – см. саму утилиту (http://openjdk.java.net/projects/code-tools/jmh/) и выступление Алексея Шипилёва (https://www.youtube.com/watch?v=8pMfUopQ9Es).

контрактов при загрузке, так и оптимального их хранения и использования во время анализа целевого приложения.

Для решения этих задач предложен метод построения графа контрактов и проверки отсутствия циклов в нём. Далее в исходный код целевого приложения с помощью техники инструментирования байт-кода встраиваются соответствующие инструкции, проверяющие наличие контракта для метода. Хранение часов осуществляется в высокоскоростной хеш-таблице по генерируемым динамически синтетическим ключам. В рамках применения этого подхода для детектора jDRD разработана архитектура модуля контрактов и его техническая реализация.

Доработанный таким образом jDRD был применён на трёх индустриальных приложениях. Измерение основных метрик показало как практическую пользу от построения графа контрактов (был выявлен ряд ошибок и несоответствий в разработанных контрактах), так и сокращение количества обрабатываемых операций синхронизации за единицу времени.

Дальнейшие работы должны включать в себя постановку полноценного эксперимента по измерению времени работы детектора – от разработки методики эксперимента до его проведения. В качестве направления дальнейшего развития средств спецификации контрактов можно указать визуальное моделирование. Интерес представляет описание иерархии контрактов и соответствующих им Java-классов и Java-методов с помощью диаграмм классов UML, расширенных и настроенных подходящим образом [36-38]. Также возможны визуальные спецификации поведения контактов с помощью динамических моделей [39]. Интересно исследовать задачу автоматизированного извлечения описания контрактов из Java-кода и Java-документации и связь повторного использования контрактов с повторным использованием документации [40-41].

Список литературы

[1] Java Language Specification, Third Edition. Threads and Locks. Happens-before Order. http://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.4.5

[2] Netzer R., Miller B. What Are Race Conditions? Some Issues and Formalizations. ACM Letters on Programming Languages and Systems, 1(1), 1992, pp. 74–88.

[3] Blackout Final Report, August 14, 2003, http://www.ferc.gov/industries/electric/indus-act/reliability/blackout/ch5.pdf

[4] Leveson N., Turner C. S. An Investigation of the Therac-25 Accidents. In IEEE Computer, vol. 26, N 7, 1993, pp. 18–41.

[5] Engler D., Ashcraft K. RacerX: Effective, Static Detection of Race Conditions and Deadlocks. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003, pp. 237–252.

[6] Voung J., Jhala R., Lerner S. RELAY: Static Race Detection on Millions of Lines of Code. In ESEC/FSE, 2007, pp. 205–214.

[7] Herlihy M., Shavit N. The Art of Multiprocessor Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008, 528 p.

[8] Kahlon V., Sinha N., Kruus E., Zhang Y.: Static data race detection for concurrent programs with asynchronous calls. In Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the Foundations of Software Engineering, 2009, pp. 13–22.

[9] Naik M., Aiken A., Whaley J. Effective Static Race Detection for Java. In Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2006, pp. 308–319.

[10] Radoi C., Dig D. Practical static race detection for java parallel loops. In Proc. of the 13th International Symposium on Software Testing and Analysis, ISSTA '13, 2013. P.178–190.

[11] Xie X., Xue J., Zhang J. Acculock: Accurate and Efficient Detection of Data Races. Softw. Practice Experience, vol. 43, no. 5, May 2013, pp. 543–576.

[12] Burckhardt S., Musuvathi M. Effective program verification for relaxed memory models. In Proceedings of the 20th international conference on Computer Aided Verification, Berlin, Heidelberg, 2008. pp. 107–120.

[13] Huynh T., Roychoudhury A. Memory model sensitive bytecode verification. Form. Methods Syst. Des., 31(3), 2007, pp. 281–305.

[14] Boyapati C., Lee R., Rinard M. Ownership types for safe programming: preventing data races and deadlocks. In Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2002, pp. 211–230.

[15] Flanagan C., Freund S. Type inference against races. Sci. Comput. Program., Vol 64, January 2007, pp. 140–165.

[16] Rose J., Swamy N., Hicks M. Dynamic inference of polymorphic lock types. Science of Computer Programming, 58(3), 2005, pp. 366–383.

[17] Biswas S., Zhang M., Bond M., Lucia B. Valor: Efficient, Software-Only Region Conflict Exceptions. In OOPSLA, 2015, pp. 241–259.

[18] Flanagan C., Freund S. FastTrack: Efficient and Precise Dynamic Race Detection. In ACM Conference on Programming Language Design and Implementation, 2009, pp. 121–133.

[19] Kini D., Mathur U., Viswanathan M. Dynamic race prediction in linear time. SIGPLAN Not. 52(6), 2017, pp. 157–170.

[20] Qi Y., Das R., Luo Z., Trotter M. MulticoreSDK: a practical and efficient data race detector for real-world applications. Proceedings Software Testing, Verification and Validation (ICST), IEEE, 21-25 March 2011, pp. 309–318.

[21] Serebryany S., Iskhodzhanov T. ThreadSanitizer: Data race detection in practice. In Proceedings of the Workshop on Binary Instrumentation and Applications, 2009, pp. 62–71.

[22] Serebryany K., Potapenko A., Iskhodzhanov T., Vyukov D. Dynamic race detection with LLVM compiler - compile-time instrumentation for ThreadSanitizer. In RV, 2011, Lecture Notes in Computer Science, vol 7186, pp. 110–114.

[23] Yu M., Bae D., SimpleLock+: Fast and Accurate Hybrid Data Race Detection. Comput. J., vol. 59, no. 6, 2016, pp. 793–809.

[24] Zhang T., Jung C., Lee D. ProRace: Practical Data Race Detection for Production Use. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017, pp. 149–162.

[25] Bond M., Coons K., McKinley K. Pacer: Proportional Detection of Data Races. Proceedings of 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2010), Toronto, June 2010, pp. 255–268.

[26] Marino D., Musuvathi M., Narayanasamy S. LiteRace: Effective Sampling for Lightweight Data Race Detection. PLDI '09 Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation, Vol. 44, Issue 6, 2009, pp. 134–143.

[27] Трифанов В.Ю. Обнаружение состояний гонки в Java-программах на основе синхронизационных контрактов. Компьютерные инструменты в образовании. №4, 2012, стр. 16-29.

[28] Трифанов В.Ю, Цителов Д.И. Динамический поиск гонок в Java-программах на основе синхронизационных контрактов. Материалы конференции "Инструменты и методы анализа программ (TMPA-2013)", Кострома, 2013, стр. 273–285.

[29] Трифанов В.Ю., Цителов Д.И. Язык описания синхронизационных контрактов для задачи поиска гонок в многопоточных приложениях. Программная инженерия. Т.8, N 6, 2017, стр. 250–257.

[30] Elmas T., Qadeer S., Tasiran S. Goldilocks: A Race and Transaction-Aware Java Runtime. Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07), 2007, pp. 245–255.

[31] Lamport L. Time, Clocks and the Ordering of Events in a Distributed System. Communications of the ACM, Vol. 21, Issue 7, 1978, pp. 558–565.

[32] Intel Thread Checker, http://software.intel.com/en-us/intel-thread-checker/

[33] Трифанов В.Ю. Динамическое обнаружение состояний гонки в многопоточных Java-программах. Дисс. на соискание степени канд. техн. наук. СПбГУ, 2013.

[34] Liskov B., Wing J. A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. 16 (6). November 1994, pp.1811–1841.

[35] Click C. A lock-free wait-free hash table. https://web.stanford.edu/class/ee380/Abstracts/070221_LockFreeHash.pdf

[36] Гаврилова Т.А., Лещева И.А., Кудрявцев Д.В. Использование моделей инженерии знаний для подготовки специалистов в области информационных технологий. Системное программирование. 2012. Т. 7. № 1, pp. 90–105.

[37] Кознов Д.В. Основы визуального моделирования. Интернет-Университет Информационных Технологий (ИНТУИТ). Москва, 2008.

[38] Ольхович Л.Б., Кознов Д.В. Метод автоматической валидации UML-спецификаций на основе языка OCL. Программирование. 2003. Т. 29. № 6, стр. 44–50.

[39] Иванов А., Кознов Д., Мурашева Т. Поведенческая модель RTST++. Записки семинара Кафедры системного программирования "Case-средства RTST++". 1998. № 1, стр. 37–52.

[40] Луцив Д.В., Кознов Д.В., Басит Х.А., Терехов А.Н. Задачи поиска нечётких повторов при организации повторного использования документации. Программирование. 2016. № 4, стр. 39–49.

[41] Кознов Д.В., Романовский К.Ю. Автоматизированный рефакторинг документации семейств программных продуктов. Системное программирование. 2009. Т. 4, стр. 128–150.

Applying synchronization contracts approach for dynamic detection of data races in industrial applications

V.Yu. Trifanov <vitaly.trifanov@gmail.com>
St. Petersburg State University,
Universitetski pr., 28, 198504 St. Petersburg, Russia

Abstract. Data race occurs in multithreaded program when several threads simultaneously access same shared data and at least of them writes. Two main approaches to automatic race detection – static and dynamic – have their pros and cons. Dynamic analysis can provide best precision on certain program execution but introduce enormous runtime overheads. Earlier we introduced high-performance approach that improves performance of dynamic race detection. The key idea is to define and exclude external trusted parts of code (e.g. libraries) from analysis and replace them with specifications of their behavior in multithreaded environment. Possible behavior was classified and corresponding language for describing contracts developed. Evaluation on lightweight applications confirmed performance boost but further industrial usage of detector revealed some problems. This article covers that problems, introduces method and architecture of contract processing module and some technical features that help to apply proposed approach on high load production systems.

Ключевые слова: multithreading; data race; dynamic analysis; automatic error detection.

DOI: 10.15514/ISPRAS-2018-30(3)-4

For citation: Trifanov V.Yu. Applying synchronization contracts approach for dynamic detection of data races in industrial applications. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 47-62 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-4

References

[1] Java Language Specification, Third Edition. Threads and Locks. Happens-before Order. http://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html#jls-17.4.5
[2] Netzer R., Miller B. What Are Race Conditions? Some Issues and Formalizations. ACM Letters on Programming Languages and Systems, 1(1), 1992, pp. 74–88.
[3] Blackout Final Report, August 14, 2003, http://www.ferc.gov/industries/electric/indus-act/reliability/blackout/ch5.pdf
[4] Leveson N., Turner C. S. An Investigation of the Therac-25 Accidents. In IEEE Computer, vol. 26, N 7, 1993, pp. 18–41.
[5] Engler D., Ashcraft K. RacerX: Effective, Static Detection of Race Conditions and Deadlocks. Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, 2003, pp. 237–252.
[6] Voung J., Jhala R., Lerner S. RELAY: Static Race Detection on Millions of Lines of Code. In ESEC/FSE, 2007, pp. 205–214.
[7] Herlihy M., Shavit N. The Art of Multiprocessor Programming. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008, 528 p.
[8] Kahlon V., Sinha N., Kruus E., Zhang Y.: Static data race detection for concurrent programs with asynchronous calls. In Proceedings of the 7th Joint Meeting of the

European Software Engineering Conference and the Foundations of Software Engineering, 2009, pp. 13–22.

[9] Naik M., Aiken A., Whaley J. Effective Static Race Detection for Java. In Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation, 2006, pp. 308–319.

[10] Radoi C., Dig D. Practical static race detection for java parallel loops. In Proc. of the 13th International Symposium on Software Testing and Analysis, ISSTA ’13, 2013. P.178–190.

[11] Xie X., Xue J., Zhang J. Acculock: Accurate and Efficient Detection of Data Races. Softw. Practice Experience, vol. 43, no. 5, May 2013, pp. 543–576.

[12] Burckhardt S., Musuvathi M. Effective program verification for relaxed memory models. In Proceedings of the 20th international conference on Computer Aided Verification, Berlin, Heidelberg, 2008. pp. 107–120.

[13] Huynh T., Roychoudhury A. Memory model sensitive bytecode verification. Form. Methods Syst. Des., 31(3), 2007, pp. 281–305.

[14] Boyapati C., Lee R., Rinard M. Ownership types for safe programming: preventing data races and deadlocks. In Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 2002, pp. 211–230.

[15] Flanagan C., Freund S. Type inference against races. Sci. Comput. Program., Vol 64, January 2007, pp. 140–165.

[16] Rose J., Swamy N., Hicks M. Dynamic inference of polymorphic lock types. Science of Computer Programming, 58(3), 2005, pp. 366–383.

[17] Biswas S., Zhang M., Bond M., Lucia B. Valor: Efficient, Software-Only Region Conflict Exceptions. In OOPSLA, 2015, pp. 241–259.

[18] Flanagan C., Freund S. FastTrack: Efficient and Precise Dynamic Race Detection. In ACM Conference on Programming Language Design and Implementation, 2009, pp. 121–133.

[19] Kini D., Mathur U., Viswanathan M. Dynamic race prediction in linear time. SIGPLAN Not. 52(6), 2017, pp. 157–170.

[20] Qi Y., Das R., Luo Z., Trotter M. MulticoreSDK: a practical and efficient data race detector for real-world applications. Proceedings Software Testing, Verification and Validation (ICST), IEEE, 21-25 March 2011, pp. 309–318.

[21] Serebryany S., Iskhodzhanov T. ThreadSanitizer: Data race detection in practice. In Proceedings of the Workshop on Binary Instrumentation and Applications, 2009, pp. 62–71.

[22] Serebryany K., Potapenko A., Iskhodzhanov T., Vyukov D. Dynamic race detection with LLVM compiler - compile-time instrumentation for ThreadSanitizer. In RV, 2011, Lecture Notes in Computer Science, vol 7186, pp. 110–114.

[23] Yu M., Bae D., SimpleLock+: Fast and Accurate Hybrid Data Race Detection. Comput. J., vol. 59, no. 6, 2016, pp. 793–809.

[24] Zhang T., Jung C., Lee D. ProRace: Practical Data Race Detection for Production Use. In Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2017, pp. 149–162.

[25] Bond M., Coons K., McKinley K. Pacer: Proportional Detection of Data Races. Proceedings of 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2010), Toronto, June 2010, pp. 255–268.

[26] Marino D., Musuvathi M., Narayanasamy S. LiteRace: Effective Sampling for Lightweight Data Race Detection. PLDI '09 Proceedings of the 2009 ACM SIGPLAN

conference on Programming language design and implementation, Vol. 44, Issue 6, 2009, pp. 134–143.

[27] Trifanov V.Yu. Detecting data races in Java programs with synchronization contracts. Komp'juternye instrumenty v obrazovanii [Computer Tools in Education]. №4, 2012, pp. 16-29. (in Russian)

[28] Trifanov V.Yu., Tsitelov D.I. Dynamic detection of data races in Java programs with synchronization contracts. Materialy konferencii "Instrumenty i metody analiza programm (TMPA-2013)" [Proc of Tools and Methods of Program Analysis conference TMPA-2013], Kostroma, 2013, pp. 273–285. (in Russian)

[29] Trifanov V.Yu., Tsitelov D.I. Language for synchronization contracts creation to detect races in multithreaded applications. Programmnaja inzhenerija [Software Engineering], vol. 8, N 6, 2017, pp. 250–257. (in Russian)

[30] Elmas T., Qadeer S., Tasiran S. Goldilocks: A Race and Transaction-Aware Java Runtime. Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07), 2007, pp. 245–255.

[31] Lamport L. Time, Clocks and the Ordering of Events in a Distributed System. Communications of the ACM, Vol. 21, Issue 7, 1978, pp. 558–565.

[32] Intel Thread Checker, http://software.intel.com/en-us/intel-thread-checker/

[33] Trifanov V.Yu. Dynamic data race detection in multithreaded Java-programs. PhD thesis, SPbSU, 2013. (in Russian)

[34] Liskov B., Wing J. A behavioral notion of subtyping. ACM Trans. Program. Lang. Syst. 16 (6). November 1994, pp.1811–1841.

[35] Click C. A lock-free wait-free hash table. https://web.stanford.edu/class/ee380/Abstracts/070221_LockFreeHash.pdf

[36] Gavrilova T.A., Leshheva I.A., Kudrjavcev D.V. Using models of knowledge engineering for growing specialists in information technologies area. Sistemnoe programmirovanie [System programming], vol. 7, № 1, 2012, pp.. 90–105. (in Russian)

[37] Koznov D.V. Basis of visual modeling. Internet-Universitet Informacionnyh Tehnologij (INTUIT) [Internet-University of Information Technologies], Moscow, 2008 (in Russian)

[38] Ol'hovich L.B., Koznov D.V. OCL-Based Automated Validation Method for UML Specifications. Programming and Computer Software, vol. 29, № 6, 2003, pp. 44–50. DOI: 10.1023/B:PACS.0000004132.42846.11

[39] Ivanov A., Koznov D., Murasheva T. Behavioral model RTST++, Zapiski seminara Kafedry sistemnogo programmirovanija "Case-sredstva RTST++" [Notes of seminar “Case-tools RTST++” of system engeneering department], 1998, № 1, pp. 37–52. (in Russian)

[40] Luciv D.V., Koznov D.V., Basit H.A., Terehov A.N. On fuzzy repetitions detection in documentation reuse. Programming and Computer Software, vol. 42, № 4, 2016, pp. 39–49. DOI: 10.1134/S0361768816040046

[41] Koznov D.V., Romanovskij K.Ju. Automated documentation refactoring for lines of program products. Sistemnoe programmirovanie [System programming], vol. 4, 2009, pp. 128–150. (in Russian)

Applying Deep Learning to C# Call Sequence Synthesis

A.E. Chebykin <a.e.chebykin@gmail.com>
I.A. Kirilenko <jake.kirilenko@gmail.com>
Faculty of Mathematics and Mechanics, Saint Petersburg State University
Universitetsky prospekt, 28, Peterhof, St. Petersburg, 198504, Russia

Abstract. Many common programming tasks, like connecting to a database, drawing an image, or reading from a file, are long implemented in various frameworks and are available via corresponding Application Programming Interfaces (APIs). However, to use them, a software engineer must first learn of their existence and then of the correct way to utilize them. Currently, the Internet seems to be the best and the most common way to gather such information. Recently, a deep-learning-based solution was proposed in the form of DeepAPI tool. Given English description of the desired functionality, sequence of Java function calls is generated. In this paper, we show the way to apply this approach to a different programming language (C# over Java) that has smaller open code base; we describe techniques used to achieve results close to the original, as well as techniques that failed to produce an impact. Finally, we release our dataset, code and trained model to facilitate further research.

Keywords: API; deep learning; code search; RNN; transfer learning.

DOI: 10.15514/ISPRAS-2018-30(3)-5

For citation: Chebykin A.E., Kirilenko I.A. Applying Deep Learning to C# Call Sequence Synthesis. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 63-86. DOI: 10.15514/ISPRAS-2018-30(3)-5

1. Introduction

When writing code, software developers often utilize various libraries via APIs. Since the problems being solved in this manner are usually similar for most users, their solutions form stable patterns of API invocations.

API mining is a long-established line of research aimed at extracting these API usage trends from source code. The importance of the task lies in the fact that generally developers spend a lot of time trying to learn frameworks' APIs in order to utilize them efficiently. A field study has found that developers often struggle to map a task from problem domain to the terminology of the API [1]. In another survey 67.6% of respondents identified that learning APIs is hindered by inadequate or absent resources [2].

Usually, when facing such problems, developers turn to general web search engines. However, those are not optimized for programming-related queries and thus tend to be inefficient [3].

An alternative lies in various approaches based on statistical analysis of source code. They can provide sequences of API methods that are often used together [4], mine API specifications in the form of automata [5], synthesize relevant code snippets [6].

Deep API Learning [7] is a recent deep learning-based take on the problem that reports state-of-the-art results. The authors formulate the problem of providing API patterns satisfying users' needs as a translation one. Input language, in which user describes desired functionality, is English, and the output language is one of API sequences: API calls are words of the language, ordered sequences of these calls form sentences. For example, English sentence "*generate random int*" could be translated to the language of Java API as "*Random.new Random.nextInt"*, which corresponds to the construction of an object of type *Random* and subsequent call of its *nextInt* method.

DeepAPI tool targets exclusively Java programming language and reportedly performs well. Benefits of the approach come from the usage of deep recurrent neural networks. Thanks to them, trained model can distinguish synonyms and impact of word sequence (for example, it can distinguish queries *convert string to int* and *convert int to string*).

However, the authors identify several threats to validity, including possible failure when extending the approach to other programming languages.

Our main goals are to test this threat, thus appraising generality of the approach, and to consider possible improvements. We choose C# as a target language due to its general similarity to Java, aiming to make a first step towards more different — and therefore challenging — target languages.

However even in our case simple copying of DeepAPI approach leads to bad results, and constructing well-working model proves to be far from trivial. In this paper, we describe our experience of extending the proposed approach to C#.

To achieve our goals we collect dataset of 2,886,309 training samples from open source projects' code and use it to first train a model with the architecture of DeepAPI (attaining the result of 10.94 BLEU), and then tune parameters to achieve BLEU 26.26. After that, we introduce data preprocessing, which reduces dataset size to 1,397,597, but improves its quality and increases BLEU metric to 46.99. Finally, we employ transfer learning on an alternative dataset of method names and achieve the best results of 50.14 BLEU, which is fairly close to the 54.42 reported by DeepAPI on Java dataset.

Additionally we ask professional developers to evaluate output of our model on several queries, which shows that on average our model, DeepAPI#, performs as well as DeepAPI.

Our main contributions are:

- reproduction of the DeepAPI experiment with a different dataset;
- modification of the approach via programming-language-independent data preprocessing which leads to results, comparable to original, despite lack of data;
- collection of C# dataset of commented methods and publishing of it for the benefit of the future research in the area;
- employment of transfer learning techniques for additional improvement of the results. To the best of our knowledge, we are the first to investigate transfer learning in the area of API mining.

The paper is organized as follows: in section 2 we outline DeepAPI model architecture. Next, in section 3 collection of the dataset needed for model training is discussed and additional preprocessing steps are introduced. We describe our application of transfer learning to the problem in section 4. Technical details of model training are reported in section 5, which is followed by section 6, where evaluation results are described. We finish the paper with section 7, where we report work done on related problems and discuss ways in which existing research differs from ours.

2. DeepAPI model

We borrow general model structure from DeepAPI, which is itself based on recent advancements in neural machine translation. Here we will provide only an overview, for details please refer to the original paper [7] and our previous research-in-progress paper [8].

Since the goal is to generate one sequence of words based on another, the task falls in the category of Sequence-to-Sequence learning [9]. One of the best architectures for the task is an Encoder-Decoder network [10].

It consists of two recurrent neural networks (Recurrent neural network is a special class of neural networks where unit can be connected to itself, thus allowing its state to serve the role of memory). Encoder network reads input sequence, Decoder generates output one. The process goes as following.

Encoder reads input word by word, embeds each one in a high-dimensional space and sequentially updates its hidden state, which by the end of the sentence contains language-independent idea of the input sentence. This state (also known as context vector) is then passed to the Decoder, which based on it and the last generated word generates words one by one until a special end-of-sequence token is outputted.

An example of such model at work can be seen in Fig. 1. In the image states of networks are rolled out in time, so for example RNN_1, RNN_2, RNN_3 is the RNN state at time steps $1, 2, 3$. Note that Encoder and Decoder consist of different RNNs and work in different time windows: at first, Encoder RNN makes 3 steps in time and then Decoder RNN makes 3 steps in time.

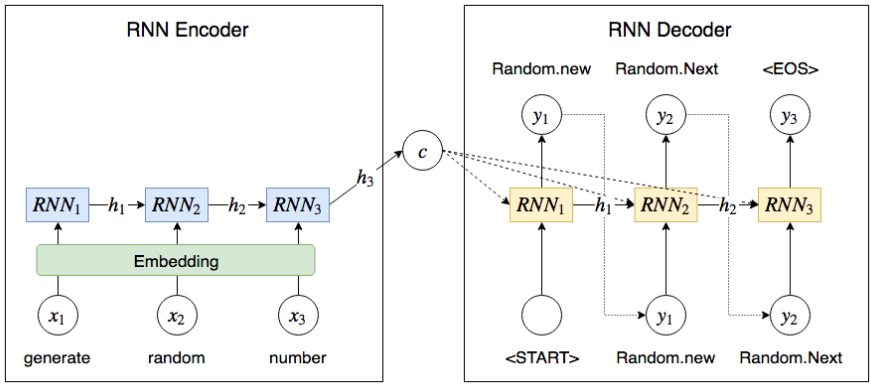


Fig. 1. RNN Encoder-Decoder workflow

The benefits of this model include synonym handling (words used in the similar contexts get embedded near each other), successful processing of long inputs thanks to the memorizing ability of the recurrent networks, and finally appreciation of word sequence impact.

One major downside of such a model is the need for a large amount of sentence pairs describing the same functionality in two languages (*"generate random number"*, "*Random.new Random.Next"*). Format of the API language description is reported in the point 3.1.3.

Source of such data can be methods' documentation comments (that in C# are XML-based and contain summary section, in which brief description of the method's functionality should be supplied) and corresponding API calls made in the method body. Details of the dataset collection are described in section 3.

There are several improvements of the Encoder-Decoder architecture that were shown to reliably improve results.

- Using Bidirectional Encoder leads to input being processed twice: in normal order and in reverse, resulting in 2 context vectors, which are then concatenated to get final context vector [11].
- Attention mechanism [12] allows decoder to focus on different input words when generating different output ones.

In the original DeepAPI paper an additional improvement is introduced in the form of a regularization term punishing generation of the most widespread and therefore probably problem-irrelevant API calls, such as logging ones. We have not tried such regularization since its reported impact on BLEU score is minimal. We leave testing of this enhancement for future research.

3. Dataset

3.1 Dataset collection

To train the model, we need to gather large amount of pairs (English description of functionality, API description). One way to do it is to process open source projects, looking for methods with documentation comments, extracting summary sections and linearizing interesting parts of ASTs (i.e. API calls). The processing of individual methods is described in section 3.1.3.

GitHub[1] is one of the most popular open source project hostings. Following DeepAPI authors, we construct our dataset from data published there.

We attempted to augment GitHub data with data from alternative sources. In our previous paper [8] we proposed using Nuget[2] – a repository of compiled C# packages. However we eventually found out that compared to GitHub it does not provide much data, and what samples it provides often duplicate ones collected from GitHub. So we discontinued using Nuget as data source.

There are other sites with published open source projects, for example, Codeplex[3] and SourceForge[4]. Unfortunately, we found there only a small amount of C# projects, many of which gradually migrate to GitHub, or have already done so. These hosting sites also lack search APIs that are essential for the automatic collection of our dataset. So the potentially small amount of additional data is nontrivial to collect, and therefore we choose to ignore these alternative sources.

We collect dataset from GitHub in several steps:

1) obtain a list of repositories relevant to us;
2) download these repositories;
3) process them, extracting from methods with documentation comments these comments, linearized in a special way API calls, types and names of method parameters.

The architecture overview can be seen in Fig. 2. Let us discuss every step in detail.

3.1.1 Obtaining list of relevant repositories

We are interested in repositories in C# language. Similar to the original paper, we would like to consider only projects that have at least one star in order to filter unused or toy projects. Both these requirements can be satisfied when setting specific parameters of GitHub Search API.

[1] github.com
[2] nugget.org
[3] archive.codeplex.com
[4] sourceforge.net

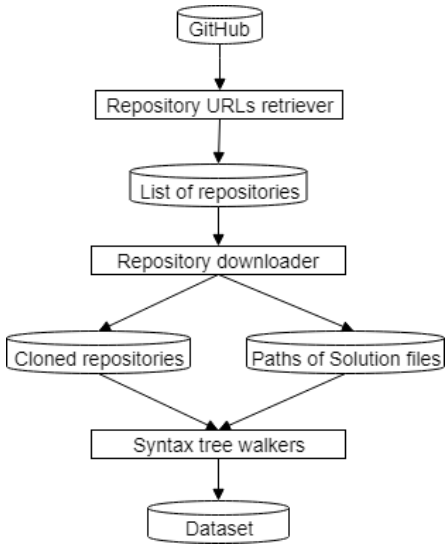


Fig. 2. Dataset gathering workflow

Using this API via Octokit.rb[5] library, we retrieve 140,990 URLs of relevant projects created from 2012 to 2017. This contrasts to the original paper that reports working with 442,928 Java repositories. Therefore, we initially have approximately 3 times less projects to work with. This lack of data can potentially be a significant obstacle when transferring the approach to other languages with smaller open code bases.

Search API also poses several technical difficulties.

Firstly, it returns no more than 1,000 results for any search request. To go around this restriction, we set additional parameter limiting repository creation date to a short span of time, for example, "2016-01-01 .. 2016-01-08". Every our requests covers 8 days, which we find short enough a period that no more than 1,000 repositories are created during it.

Secondly, Search API limits number of requests per minute by 30. In order not to exceed this limit, our script sleeps for 2 seconds after each request.

We store repositories list and the rest of our data in a SQLite database[6].

3.1.2 Downloading repositories

Having gathered repository list, we can start cloning them with git. We set clone depth to 1 to speed up the process.

[5] github.com/octokit/octokit.rb
[6] sqlite.org

After download, we search for solution files — special files that encompass source code files, as well as store project dependencies. We process these files in the next step.

3.1.3 Extracting data

C# type system is problematic for our purposes compared to Java because of the implicit type "var" introduced in version 3.0. As a consequence of its existence, code needs to be compiled in order for the type of a variable to be determined correctly, as opposed to Java where name of the variable's type or supertype is evident from its declaration. This need for compilation limits number of projects we can process.

For compilation and syntax tree processing, we use Roslyn[7] — an open source C# compiler developed by Microsoft. To compile a project we need it to satisfy two requirements:

1) no manual actions are necessary for its build and compilation;
2) a solution file, encompassing source code files, must exist.

In order to compile more projects, we employ Nuget to restore project dependencies prior to compilation.

About 80.6% percent of repositories contain solution files, and of those 47.1% could be compiled.

After compilation, we process projects in the following fashion:

1) find methods with documentation comments;
2) store whole comment and summary section;
3) walk syntax tree of the method body, collecting API call sequence;
4) store method name;
5) store parameter types and names, which we think can potentially provide valuable information, but are not used in this work.

An example of extracting data from method with documentation comment is provided in Fig. 3.

We construct API sequence similarly to the original paper. We traverse the tree in the way an interpreter might traverse it during execution, e.g. depth-first post order, processing method call's arguments before processing the call itself, and so on. When encountering constructor invocation *new C()*, we add *C.new* to the API sequence. When encountering method call *o.m()* where *o* is an instance of a class C, we add *C.m* to the API sequence. Additionally, when encountering *if-else* statement, we firstly process condition expression, then *if*-branch statements and finally *else*-branch statements.

We introduce one additional step to this scheme: when encountering *try-with-resources* node, we save the class *C* of an object being created in the *try* node and

[7] github.com/dotnet/roslyn

after processing everything inside *try* branch we add *C.Dispose* to the API sequence. While it is easier for a programmer to rely on the language feature of *try-with-resources* block to take care of finalization of the resources, this construct is not always used, and we think that our model should know that certain sequences of API calls end with finalization call.

Eventually we obtain 2,886,309 pairs of English descriptions and API sequences. However, this number is not directly comparable to the 7,519,907 methods reported in the DeepAPI paper. The authors explained to us (in an e-mail) that 7,519,907 is the amount of data after filtering out-of-vocabulary words, the step which in our experience removes certain samples entirely, significantly reducing size of the dataset.

Our preprocessing and the final size of dataset is discussed in the further section.

```
/// <summary>
/// Copies the data from one stream to another.
/// </summary>
/// <param name="from">The source stream.</param>
/// <param name="to">The destination stream.</param>
private void Copy(Stream from, Stream to)
{
    var reader = new StreamReader(from);
    var writer = new StreamWriter(to);
    writer.WriteLine(reader.ReadToEnd());
    writer.Flush();
}
```

```
Comment description: copies the data from one stream to another
API calls: System.IO.StreamReader.new System.IO.StreamWriter.new
           System.IO.StreamReader.ReadToEnd
           System.IO.StreamWriter.WriteLine System.IO.StreamWriter.Flush
Full name: PDS.WITSMLstudio.Desktop.Core.Providers.LoggingSoapExtension.Copy
Tokenized name: logging soap extension copy
Parameters: System.IO.Stream from, System.IO.Stream to
```

Fig. 3. Example of data extraction

3.2 Data preprocessing

Upon inspecting the gathered data we conclude that it can be improved prior to being used for model training. By introducing following preprocessing steps we aim to make the training easier and the results consequently better - a notion supported by our experiments (see section 6).

3.2.1 Language detection

We consider our model to work with English language as input, however, many comments are not in it. Therefore, we try to filter out non-English comments using a language detection package[8].

We find, however, that some English sentences are recognized as non-English. In our opinion, most likely reasons are extreme shortness of sentences used for language detection and uncommon profession-specific programmers' vocabulary. We do not want to decrease dataset size by filtering out comments incorrectly recognized as non-English, and so we change our filtering approach.

Instead of leaving only sentences recognized as English, we remove ones that are reported to be in a set of well-recognizable languages (which we deduce by hand examination) that occur in our dataset most often. Languages, sentences in which we remove, are Chinese, Korean, Japanese, Russian, German and Polish (reported in the order of decreasing frequency). As a side note, the reason for good recognition of said languages probably lies in them having alphabets different from the English one.

Such filtering leads to vocabulary containing mostly English words. It reduces training size from 2,886,309 pairs to 2,606,424.

3.2.2 Leaving only distinct pairs

The percent of unique pairs is about 86.6%. Note that we consider two pairs distinct even if English descriptions coincide while API descriptions do not, and vice versa.

We could identify several reasons for occurrence of repetitions:

- auto-generated code and comments (Windows Forms are especially ubiquitous);
- libraries being copied to the project sources instead of being linked as dependencies.

This step reduces amount of training instances from 2,606,424 to 2,259,653.

3.2.3 Repetition contraction

In some API sequences an API call is repeated several times in a row. This could happen as a result of our AST linearization in a situation where, for example, an API call is made with different parameters in branches of an *if-else* statement. Since we do not record call parameters and when linearizing *if-else* statement save API calls from both branches, this may lead to an API call repeating twice in the resulting sequence. End user would not care about such repetitions in the output of the model, so we remove them before training, leaving only one copy of API call in a row.

This step does not influence amount of data, but rather is intended to improve quality of the existing training samples

[8] github.com/Mimino666/langdetec

3.2.4 Vocabulary filtering

Similar to the original paper, we create vocabularies of 10,000 most popular words in each language, and filter out the rest. If after filtering no words are left in either English description or API one, we remove the pair altogether.

This step reduces training dataset size significantly, from 2,259,653 to 1,397,597.

3.2.5 Stemming

Additionally we experiment with, but eventually discard a preprocessing step of stemming.

Stemming is the process of reducing inflected words to their bases. We intended to use it, as is usual, to decrease vocabulary by replacing multiple word forms with the root.

In our case it fails to provide improvement and instead makes results worse. A possible explanation may lie in the fact that stemming model was trained on regular words, not ones specific for software development and therefore works badly with this unusual vocabulary.

We discuss impact of the preprocessing steps in section 6.

The final size of our dataset is 1,397,597 pairs, which is more than 5 times smaller than 7,519,907 pairs used for training in the original paper. Even if only preprocessing from the original paper is used (i.e. vocabulary filtering and nothing else), dataset size is 1,692,898 (of which 1,434,805 pairs are unique). We consider this a significant problem that very probably makes achieving comparable results harder and takes a great toll on the model performance.

For easy reproduction of our research and for conduction of new experiments in the area, we provide our dataset[9], as well as the code used to collect[10] and preprocess[11] it.

4. Transfer learning for API mining

Broadly speaking, transfer learning is utilizing knowledge gained in one problem to solve another. It is often used in NLP [13] and neural machine translation, especially in the contexts where data is scarce [14]. Since our situation is one of lacking data (as shown by an experiment in section 6), we decided to investigate this idea.

4.1 Alternative dataset

To apply transfer learning to our problem of generating API calls given English description, we need to train a model for a task that is different, yet very similar.

As already mentioned, the DeepAPI paper proposes method body as a source of API description of the functionality and method comment as a source of the English one. But there is another description for a method functionality beside its comment — its

[9] kaggle.com/awesomelemon/csharp-commented-methods-github
[10] github.com/AwesomeLemon/api-extraction
[11] github.com/AwesomeLemon/api-extraction-scripts

name. Combined with class name, it seems descriptive of the method's contents. While not forming proper natural language sentences, these names could provide crude approximations.

Examples of correspondence between comments and names of the methods are provided in Table 1. It can be seen that generally tokenized names are very similar to summary sections of documentation comments. However, this is not always the case. In the last two examples despite similarity between comment and name, essential information is missing from the tokenized name. In the first of these samples key word is "Matches", without it tokenized method name loses meaning. In the second one "Dword" is separated to "d" and "word" due to the tokenizing technique. When we tokenize method name, we assume that naming guidelines are followed and therefore first letter of the method name and first letters of every word in the name are capitalized. Here this leads to a wrong division of words and thus vital information disappears, making description senseless.

However, in most cases method names tokenized in this way are similar to comments and thus provide relatively good description of method contents.

We start exploration of this alternative dataset by simply training a model on it with the best parameters and our preprocessing. Results are not very good (model №4 in Table 2; the table is discussed minutely in section 6).

We conclude that comments indeed seem to be more descriptive of method contents than method names. But can we utilize this new dataset nonetheless?

Table 1. Comparison between method names and comments

Full method name	Tokenized method name	Summary section of documentation comment
Method name corresponds to comment well		
ManagedFusion. Serialization. JsonSerializer.Serialize	json serializer serialize	Serializes to JSON
MathNet.Symbolics. Packages.Standard. Structures. ComplexValue.Cosine	complex value cosine	Trigonometric Cosine of an angle in radians
StickyDesk. Utilities.ResizeBitmap	utilities resize bitmap	Resizes a bitmap image
Nini.Config. IniConfigSource. RemoveSections	ini config source remove sections	Removes all INI sections that were removed as configs
Method name corresponds to comment badly		
Spark.Parser. CharGrammar. StringOf	char grammar string of	Matches a string of characters
TagLib.Asf. DescriptionRecord. ToDWord	description record to d word	Gets the DWORD value contained in the current instance.

4.2 Applying transfer learning for model improvement

We hypothesize that the alternative method names dataset contains valuable information about correspondence between English words and API calls.

In terms of transfer learning, we can consider both our source task and target task to be the same, namely to generate API call sequence given English description of it. The difference lies in the datasets. When training for the source task, we can use the alternative dataset of pairs (Tokenized method name, API call sequence). Then we can utilize gained knowledge when training the model for the target task, which makes use of the original dataset of pairs (Documentation comment summary, API call sequence).

Therefore, we train a model on the alternative dataset, and then use learned weights for initializing the model to be trained on the standard dataset, which is a technique known as pretraining.

In addition, we wonder if we can similarly bootstrap learning without using an alternative dataset. We perform an experiment by training the model on the comments dataset and using it for initialization and training on the same dataset.

We evaluate impact of both approaches in section 6.

5. Model training

Per description in section 2, original authors use Encoder-Decoder architecture. As implementation of RNN they choose GRU [10]. They use 1-layered model with 1,000 hidden units and 120 dimensions for word embedding. To train the model, GroundHog[12] is used.

GroundHog since then has been discontinued, instead we use popular modern framework OpenNMT [15] that is designed specifically to train neural translation models.

We start training from the architecture reported in the original paper. After getting bad results we go on and empirically tune parameters, eventually arriving at following values. As RNN implementation we use LSTM [16] — a more complex model than GRU, with on-par performance, which is highly dependent on the problem. In our task it performs better. We find that 1 layer makes model not complex enough to work with C#, and since it is known that adding more layers increases model's learning ability [17], we introduce additional layers to the total of 3, which impacts results positively. We leave number of hidden units at 1,000 and word embedding at 120 dimensions.

For training, Stochastic Gradient Descent [18] is used with batch size of 32 and exponential learning rate decay. We initialize learning rate to be 1.0 and start multiplying it by 0.7 after every epoch, starting from the sixth one. Every model is trained for approximately 25 epochs on the server equipped with one Nvidia GTX 1070 GPU.

[12] github.com/pascanur/GroundHog

For model testing we separate 12,000 random pairs of descriptions from the dataset; the rest is used for training. We publish our trained model for easy reproduction of the results[13].

After training, when translating queries to API sequences we follow original authors in using beam search [19], a heuristic search algorithm popular in statistical translation. Instead of generating only the most probable word on every step, we generate multiple, and then keep only several most probable sequences. This approach solves the problem of discarding good translation sequences because of some sub-optimal words.

6. Evaluation

6.1 Metrics

In the area of API mining there are no universally adopted metrics. For better comparison to the original paper we follow in its steps and calculate BLEU score [20] for intrinsic evaluation, FRank score [6] and Precision@N for extrinsic one.

6.1.1 BLEU

BLEU is a standard metric used in machine translation to evaluate how closely generated translation resembles reference one. It does not consider grammar or others high-level features, instead calculating corrected geometric mean of n-gram precision on the whole test set [20].

Since we expect the model to generate sequences of API calls similar to the ones extracted from human-written source code, n-gram approach is applicable to our situation. The theoretical foundations of the metric stand in our case, despite target language being language of API calls rather than natural language.

BLEU is reported on the scale from 0 to 100, where higher score corresponds with bigger similarity between generated and reference sequences.

6.1.2 FRank

FRank metric value is the position of the first relevant result in the ranked list, as decided by a human evaluator. Such a metric is justified by two facts. Firstly, good scores of it show that the model has solved exactly the problem we intended for it, i.e. the problem of translating from English to relevant API calls. It was possible for the model to learn a target function uninteresting for us, in which case human evaluators would not find in model output API calls, relevant to the input.

Secondly, it is known that humans scan through ranked results from top to bottom [21], thus making it a desired trait for a model to rank relevant output higher.

[13] public-resources.ml-labs.aws.intellij.net.s3.amazonaws.com/deep-api-sharp/deep-api-sharp-model.t7

In our case FRank is measured on the scale from 1 to 10 (since similar to the DeepAPI paper, our model generates 10 outputs for every query), where lower is better. Where models fail to provide relevant results, FRank is considered to be 11.

6.1.3 Precision@N

Precision@N measures percentage of the relevant results in the first N outputs produced by the system. Following DeepAPI, we report Precision@5 and Precision@10 (note that the term used in the DeepAPI paper is "relevancy ratio N", which does not seem to be an established term).

This metric is reported on the scale from 0 to 100, where higher is better.

6.2 BLEU evaluation

In Table 2 we report results of our experiments in terms of BLEU score. We start experiments with model architecture reported in the original paper and achieve surprisingly bad results of 10.94 BLEU, which is significantly worse than 54.42 BLEU reported in the paper. Since Java and C# are fairly similar, we expected original model to work better. Possible explanation may lie in the size of our dataset, which is more than 5 times smaller.

Table 2. BLEU scores for various models

№	Parameters	Dataset	Preprocessing	Transfer learning from model №	BLEU
Parameter tuning					
1	original	comments	-	-	10.94
2	tuned	comments	-	-	26.26
Data preprocessing					
3	tuned	comments	yes	-	46.99
Different datasets					
4	tuned	names	yes	-	28.57
5	tuned	comments (part)	yes	-	36.63
6	tuned	comments and names	yes	-	44.31
Transfer learning					
7	tuned	comments	yes	3	46.18
8	tuned	comments	yes	4	**50.14**

Model with tuned parameters achieves higher BLEU score of 26.26, which is still far from the original results.

After introduction of our preprocessing steps a 94% increase in BLEU is obtained, and the resulting score of 46.99 comes fairly close to the reported performance of the DeepAPI model.

The best result is achieved by model №8, where we employ transfer learning techniques and pretrain the model on the alternative dataset of method names. Additional analysis of transfer learning application is presented in section 6.3.

Model №4 was trained on the alternative dataset of method names (with the size of 1,967,414 pairs) and yielded not outstanding BLEU score of 28.57. So our model performs worse on the alternative dataset, which is logical, given that descriptions there are not in grammatically correct English and sometimes do not provide good descriptions of functionality, as already discussed in section 4.

To measure if number of training instances indeed impacts model result, as we hypothesized, we try to train the model on 800,000 samples as opposed to the usual 1,397,597. This is the model №5, and it achieves 36.63 BLEU, which is worse than 46.99 achieved under the same parameters, but bigger dataset size. This leads us to the conclusion that dataset size is vital for model performance.

6.3 Transfer learning evaluation

We ask several questions regarding our application of transfer learning techniques:

1) Does it improve our results?
2) Can we use the model itself for pretraining, without utilizing model trained on the alternative dataset?
3) Is transfer learning necessary for performance improvement, or are instead our two datasets so similar that they could be merged and considered as one big dataset?

We answer these questions with several experiments, and come up with following answers.

1) Transfer learning leads to the best results achieved by us (model №8 with BLEU score of 50.14).
2) A model with sub-optimal parameters (which we do not include in the table in order not to clutter it) is improved by approximately 2.5 BLEU when pretrained on itself. However, best model is not, as shown by performance of model №7, that achieves only 46.18 BLEU, which approximately equals the result of the model №3 used for pretraining. So bootstrapping with the dataset itself may make sense sometimes, but not always. Presumably, model №3 was trained so well that there was no room for improvement.
3) We try to merge comments and names in one dataset, which we use for training model №6. Resulting BLEU score of 44.31 is better than using only names (28.57 BLEU, model №4), but worse than using only comments (46.99 BLEU, model №3). Thus we conclude that datasets are fairly different and should not be used together in a straightforward way.

6.4 Human evaluation

DeepAPI reports FRank, Precision@5, Precision@10 on two types of queries: popular ones, that often occur in Bing search log [6], and ones designed to showcase abilities of the proposed approach, including handling of semantically similar requests, longer input handling, combination of several tasks.

We would like to address a potential problem in evaluating the model on queries from the first group. While the DeepAPI paper reports that they do not occur in the training dataset, it seems unlikely since they were chosen for the perk of being popular, i.e. widespread, and authors do not mention filtering them out.

We test the hypothesis that such popular queries occur in the dataset by searching for them in ours. In our training data most of these popular inputs occur multiple times as exact matches. For example, “copy file” occurs 14 times, “reverse string” occurs 7 times, “execute sql statement” occurs 14 times. We conduct this search after filtering out non-unique pairs, so for these occurrences API calls do not coincide; however, they are very similar. Therefore, we believe that testing on such inputs makes little sense, because it essentially means testing on the training set, which speaks only about the model’s ability to memorize. That is expected from any model, and consequently is not very interesting.

However, to show that our model is capable of that, we test on 5 of these queries (the first 5 queries in Table 3).

However, more interesting is the inspection of the model’s ability to generalize, i.e. use gained knowledge to work with novel data. The model should be able to handle combined or semantically similar requests that are not included in the training data. We evaluate our model on 4 new queries, constructed for this exact purpose, and one such query from the DeepAPI paper. Since DeepAPI paper does not report results on 4 new queries, we used online demo of the tool[14] to generate corresponding sequences.

To avoid conflict of interest, we ask 5 professional developers to evaluate extrinsic metrics for our model. Since the correspondence between query and model output is viewed differently by every developer and is up to debate, we consider relevant only those answers that were marked as relevant by at least 2 developers.

In the Table 3 we report results of extrinsic evaluation. In general our model performs approximately the same as the original, which, having established importance of data and our lack of it, we consider an achievement.

Table 3. Extrinsic model evaluation

Query	DeepAPI			DeepAPI#			DeepAPI# output
	FRank	P@5	P@10	FRank	P@5	P@10	
convert int to string	2	40	90	1	80	50	CultureInfo.InvariantCulture Int64.ToString

[14] 211.249.63.55

convert string to int	1	100	100	3	40	60	Int32.TryParse
get current time	10	10	10	1	60	40	DateTime.Now
get files in folder	3	40	50	1	80	90	DirectoryInfo.GetFiles FileInfo.Name List.Add
generate md5 hash code	1	100	100	1	80	60	MD5.Create Encoding.GetBytes MD5.ComputeHash Byte[].Length StringBuilder.Append StringBuilder.ToString
copy a file and save it to a destination path	1	100	100	2	40	40	File.Exists String.Equals File.Exists IO.File.Copy
create socket and then send text	1	100	90	3	20	10	AddressFamily.InterNetwork SocketType.Stream ProtocolType.Tcp Socket.Connect Encoding.GetBytes Socket.Send SocketShutdown.Both Socket.Shutdown Socket.Close
write text using socket	-	0	0	1	100	100	ASCIIEncoding.GetBytes Socket.Send
connect to database and execute statement	1	80	50	6	0	30	IDbConnection.Open IDbConnection.CreateCommand IDbCommand.CommandText IDbCommand.ExecuteScalar Convert.ToInt32 Exception.ToString Console.WriteLine IDbConnection.Close
download from url and save image	3	20	20	1	60	50	String.IsNullOrEmpty WebClient.DownloadFile
Average scores	3.4	59	61	2.0	56	52	

Our model produces slightly less amount of relevant outputs (as shown by Precision@N scores), but ranks these outputs slightly higher (as shown by FRank).

Good performance on the first 5 queries demonstrate that our model is capable or memorizing correct answers, and outputs to the second 5 queries show that it can manage long requests, that require performing several action, as well as semantically similar requests.

However, both models are not very stable to slight semantic variations in the input. For example, query “create socket and then send text” is understood very well by

DeepAPI, while DeepAPI# produces low amount of relevant answers, and on the contrary, query "write text using socket" perplexes DeepAPI, that generates no socket-related calls in the top 10 results, while DeepAPI# generates only relevant output.

Additionally it should be noted that while models' outputs are not directly comparable due to different target languages, both models should still be able to correctly answer queries we are testing them on, since these tasks are fairly common and programming-language-independent.

6.5 Limitations

As already discussed in the previous section, our model can be inconsistent and sensitive to query wording. While DeepAPI# is capable of understanding synonymous queries and generating similar relevant output, it does not generate exactly the same sequences.

In addition, our model is data-hungry. While we do not artificially limit our vocabulary with standard C# library, as DeepAPI does with Java and JDK, we still observe that the model cannot take into account APIs with low amount of usages. It can work with extremely popular Math.NET and Json.NET, but not with many other frameworks, even though their APIs are included in the model dictionary. It remains an open problem for the further research to find ways to make model less data-hungry, or to fine-tune it for use of specific not very popular libraries.

7. Related work

7.1 API usage pattern mining

This group of projects is primarily concerned with extracting common usages of the library. The first algorithm to mine API patterns was MAPO [22]. It starts with clustering API sequences, then for every cluster finds API calls that are the most frequent there and passes those to an API usage recommender, that ranks API calls according to their similarity to the code context.

UP-Miner [23] improves upon MAPO by using API call sequence n-grams as a clustering metric and an additional clustering step. A near parameter-free approach PAM [4] significantly outperforms both MAPO and UP-Miner, introducing a probability model constructed in the form of a joint probability distribution over API calls observed in code and the underlying unobserved API patterns, used by developer. *Acharya et al.* [24] extract API patterns as partial orders, and unfortunately do not compare results to those of previous approaches.

The differences of these projects from our work are twofold. Firstly, these models do not allow user to specify their exact needs (MAPO and UP-Miner take API call as input, but an API call can be utilized in more than one scenario, therefore using it as input can be ambiguous; PAM and framework of *Acharya et al.* do not ask for input). This leads to the output containing many samples irrelevant to user, while not

guaranteeing to provide those he was wishing for. Secondly, to use such models one needs to know beforehand which API calls (in case of MAPO and UP-Miner) or libraries (in case of PAM and framework of *Acharya et al.*) he is interested in. Our approach allows for recommendation of APIs to use, as well as the specifics of the usage.

7.2 Generating source code from natural language

Code generation based on natural language input is one of the holy grails of Computer Science. It could be seen as a more promising alternative to our problem: after all, rather than generate API call sequence and leave it to the software developer to write code utilizing it, it would be better to just generate code in the first place.

However, current research in the area seems to be far from this dream. It mostly focuses on Domain Specific Languages [25], [26], which are simpler than general-purpose programming languages and have by definition limited usage domain.

Recent developments in generating code in general-purpose languages include works by *Ling et al.* [27] and *Yin et al.* [28]. The first paper proposes a novel approach of Latent Predictor Networks that allows for better copying of relevant key words from input to output. The second paper introduces a special version of Encoder-Decoder model, where Decoder is tailored to generate syntax trees as opposed to sequences.

The main difference between these works and ours lies in the datasets. *Ling et al.* and *Yin et al.* report results on two datasets: code of Hearthstone cards and annotated Django code (*Ling et al.* also report results on the dataset of code of Magic the Gathering cards, but this dataset is semantically very similar to the Hearthstone one). The target code for the Hearthstone dataset is rather homogenous and limited to small subset of the wide variety that is the Python language, thus resembling code in DSL more than code in general-purpose language. And while Django dataset covers various usage scenarious, it contains impractically sesquipedalian natural language descriptions of every line of code. For example description of the line "`for i in range(0, len(result)):`" is "for every i in range of integers from 0 to length of result, not included". The generation of code from descriptions several times longer than itself seems impractical.

Our dataset, on the other hand, contains wide variety of API usages, described by reasonably long sentences like "Serializes to JSON", which resemble real queries written by programmers in order to look up interesting APIs.

7.3 Deep neural machine translation and source code

Deep API Learning paper [7] itself was published in 2016, is widely cited, but little work has followed from it. The authors went on to successfully apply the neural machine translation approach to code migration between Java and C# [29], which shows that the proposed architecture can model both languages of API sequences well.

Lin et al. [30] similarly to us apply the Encoder-Decoder approach to a different target language, specifically Bash. They succeed, but it should be noted that their research problem is a simpler one in terms of target language used, since only 17 commands were selected from Bash. Together with command flags, types of open-vocabulary constants and logical connectives (&&, ||, parentheses) total output dictionary size does not exceed 300. To contrast that, our work is concerned with the same API dictionary size as original paper, which is 10,000 and therefore requires vastly bigger dataset and more complex model.

8. Conclusion

In this paper, we applied deep learning approach for recommendation of C# API calls, removing one of the threats to the validity of the paper that originally proposed this approach for Java. To achieve this goal, we collected massive dataset, introduced several data preprocessing steps, and finally employed transfer learning techniques.

Extending DeepAPI approach turned out to be nontrivial even for a similar language. Nonetheless, its main idea of modelling API sequences with RNN Encoder-Decoder stands.

Data preprocessing steps, suggested by us, are not dependent on C# and should therefore be applicable to any programming language, thus they should make extending the approach even to very different languages much easier.

By releasing data, code and trained model we hope to allow for repeatability of the experiments and to inspire further research in the area.

Acknowledgment

The authors would like to thank JetBrains Research[15] for providing a GPU-equipped server for fast machine learning models training, as well as for the Young Researcher stipend granted to our team. Additionally we would like to thank Kirsanov Alexander and other friendly developers from the JetBrains ReSharper team for their input in evaluating FRank and Precision@N metrics.

References

[1]. M. P. Robillard and R. Deline. A field study of api learning obstacles. Empirical Software Engineering, vol. 16, no. 6, 2011, pp. 703–732.

[2]. M. P. Robillard. What makes apis hard to learn? Answers from developers. IEEE software, vol. 26, no. 6, 2009, pp, 27-34.

[3]. J. Stylos and B. A. Myers. Mica: A web-search tool for finding api components and examples. In Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2006, pp. 195–202.

[4]. J. Fowkes and C. Sutton. Parameter-free probabilistic api mining across github. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 254–265.

[15] research.jetbrains.org

[5]. S. Shoham, E. Yahav, S. J. Fink, and M. Pistoia. Static specification mining using automata-based abstractions, IEEE Transactions on Software Engineering, vol. 34, no. 5, 2008, pp. 651–666.
[6]. M. Raghothaman, Y. Wei, and Y. Hamadi. Swim: Synthesizing what i mean-code search and idiomatic snippet synthesis, In Proc. of the IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 357–367.
[7]. X. Gu, H. Zhang, D. Zhang, and S. Kim. Deep api learning In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 631–642.
[8]. A. Chebykin, M. Kita, and I. Kirilenko. Deepapi#: Clr/c# call sequence synthesis from text query. In Proceedings of the Second Conference on Software Engineering and Information Management, vol. 1864. CEUR-WS.org, 2017, pp. 6–11. (in Russian) [Online]. Available: http://ceur-ws.org/Vol-1864/paper 5.pdf
[9]. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, 2014, pp. 3104–3112.
[10]. K. Cho, B. Van Merri¨ enboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078, 2014.
[11]. M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, vol. 45, no. 11, 1997, pp. 2673–2681.
[12]. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.
[13]. P. H. Calais Guerra, A. Veloso, W. Meira Jr, and V. Almeida. From bias to opinion: a transfer-learning approach to real-time sentiment analysis. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, 2011, pp. 150–158.
[14]. B. Zoph, D. Yuret, J. May, and K. Knight. Transfer learning for low-resource neural machine translation. arXiv preprint arXiv:1604.02201, 2016.
[15]. G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. arXiv preprint arXiv:1701.02810, 2017.
[16]. F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. Neural Computation, vol. 12, issue 10, 2000, pp. 2451-2471
[17]. A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In Proceedings of the IEEE international conference on Acoustics, speech and signal processing, 2013, pp. 6645–6649.
[18]. J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics, vol. 23, 1952, pp. 462–466.
[19]. P. Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In Proceedings of the Conference of the Association for Machine Translation in the Americas, 2004, pp. 115–124.
[20]. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting on association for computational linguistics, 2002, pp. 311–318.
[21]. L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, 2004, pp. 478–479.
[22]. T. Xie and J. Pei. Mapo: Mining api usages from open source repositories. In Proceedings of the 2006 international workshop on mining software repositories, 2006, pp. 54–57.

[23]. J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. Mining succinct and high-coverage api usage patterns from source code. In Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 319–328.
[24]. M. Allamanis and C. Sutton. Mining idioms from source code. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 472–483.
[25]. A. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, S. Roy. Program synthesis using natural language. In Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 345–356.
[26]. S. Gulwani and M. Marron. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 2014, pp. 803–814.
[27]. W. Ling, E. Grefenstette, K. M. Hermann, T. Koˇ cisk` y, A. Senior, F. Wang, and P. Blunsom. Latent predictor networks for code generation. arXiv preprint arXiv:1603.06744, 2016.
[28]. P. Yin and G. Neubig. A syntactic neural model for general-purpose code generation. arXiv preprint arXiv:1704.01696, 2017.
[29]. X. Gu, H. Zhang, D. Zhang, and S. Kim. Deepam: Migrate apis with multi-modal sequence to sequence learning. arXiv preprint arXiv:1704.07734, 2017.
[30]. X. V. Lin, C. Wang, D. Pang, K. Vu, and M. D. Ernst. Program synthesis from natural language using recurrent neural networks. Technical Report UW-CSE-17-03-01, University of Washington, Department of Computer Science and Engineering, 2017.

Применение глубокого машинного обучения к синтезу цепочки вызовов C#

А.Е. Чебыкин <a.e.chebykin@gmail.com>
Я.А. Кириленко <jake.kirilenko@gmail.com>
Математико-механический факультет,
Санкт-Петербургский государственный университет
Университетский пр., дом 28, Санкт-Петербург, 198504, Россия

Аннотация. Большая часть стандартных для программирования задач — например, соединение с базой данных, отображение картинки, чтение файла — давно реализована в различных библиотеках и доступна через соответствующие Application Programming Interfaces (APIs). Однако чтобы воспользоваться ими, разработчик должен сначала узнать, что они существуют, а затем — как правильно с ними работать. В настоящее время Интернет кажется наилучшим и самым популярным источником подобной информации. Недавно был предложен другой подход, основанный на глубоком машинном обучении и реализованный в виде инструмента под названием DeepAPI. По описанию желаемой функциональности на английском языке он генерирует цепочку вызовов Java функций. В данной статье мы показываем, как подход может быть перенесен на другой язык программирования (C# вместо Java), на котором доступно меньше открытого кода; мы описываем техники, позволившие достичь результата, близкого к оригинальному, а также техники, которые не улучшили производительность.

Наконец, чтобы облегчить будущие исследования в области, мы публикуем наши набор данных, код и обученную модель.

Ключевые слова: API; глубокое обучение; поиск кода; рекуррентная нейронная сеть; обучение с подкреплением.

DOI: 10.15514/ISPRAS-2018-30(3)-5

Для цитирования: Чебыкин А.Е., Кириленко Я.А. Применение глубокого машинного обучения к синтезу цепочки вызовов C#. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 63-86 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-5

Список литературы

[1]. M. P. Robillard and R. Deline. A field study of api learning obstacles. Empirical Software Engineering, vol. 16, no. 6, 2011, pp. 703–732.
[2]. M. P. Robillard. What makes apis hard to learn? Answers from developers. IEEE software, vol. 26, no. 6, 2009, pp, 27-34.
[3]. J. Stylos and B. A. Myers. Mica: A web-search tool for finding api components and examples. In Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing, 2006, pp. 195–202.
[4]. J. Fowkes and C. Sutton. Parameter-free probabilistic api mining across github. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 254–265.
[5]. S. Shoham, E. Yahav, S. J. Fink, and M. Pistoia. Static specification mining using automata-based abstractions, IEEE Transactions on Software Engineering, vol. 34, no. 5, 2008, pp. 651–666.
[6]. M. Raghothaman, Y. Wei, and Y. Hamadi. Swim: Synthesizing what i mean-code search and idiomatic snippet synthesis, In Proc. of the IEEE/ACM 38th International Conference on Software Engineering (ICSE), 2016, pp. 357–367.
[7]. X. Gu, H. Zhang, D. Zhang, and S. Kim. Deep api learning In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 631–642.
[8]. A. Chebykin, M. Kita, and I. Kirilenko. Deepapi#: Clr/c# call sequence synthesis from text query. In Proceedings of the Second Conference on Software Engineering and Information Management, vol. 1864. CEUR-WS.org, 2017, pp. 6–11. (in Russian) [Online]. Режим доступа: http://ceur-ws.org/Vol-1864/paper 5.pdf
[9]. I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Advances in neural information processing systems, 2014, pp. 3104–3112.
[10]. K. Cho, B. Van Merri¨ enboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078, 2014.
[11]. M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, vol. 45, no. 11, 1997, pp. 2673–2681.
[12]. D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473, 2014.
[13]. P. H. Calais Guerra, A. Veloso, W. Meira Jr, and V. Almeida. From bias to opinion: a transfer-learning approach to real-time sentiment analysis. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, 2011, pp. 150–158.

[14]. B. Zoph, D. Yuret, J. May, and K. Knight. Transfer learning for low-resource neural machine translation. arXiv preprint arXiv:1604.02201, 2016.
[15]. G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. Opennmt: Open-source toolkit for neural machine translation. arXiv preprint arXiv:1701.02810, 2017.
[16]. F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. Neural Computation, vol. 12, issue 10, 2000, pp. 2451-2471
[17]. A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In Proceedings of the IEEE international conference on Acoustics, speech and signal processing, 2013, pp. 6645–6649.
[18]. J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. The Annals of Mathematical Statistics, vol. 23, 1952, pp. 462–466.
[19]. P. Koehn. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In Proceedings of the Conference of the Association for Machine Translation in the Americas, 2004, pp. 115–124.
[20]. K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In Proceedings of the 40th annual meeting on association for computational linguistics, 2002, pp. 311–318.
[21]. L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, 2004, pp. 478–479.
[22]. T. Xie and J. Pei. Mapo: Mining api usages from open source repositories. In Proceedings of the 2006 international workshop on mining software repositories, 2006, pp. 54–57.
[23]. J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. Mining succinct and high-coverage api usage patterns from source code. In Proceedings of the 10th Working Conference on Mining Software Repositories, 2013, pp. 319–328.
[24]. M. Allamanis and C. Sutton. Mining idioms from source code. In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp. 472–483.
[25]. A. Desai, S. Gulwani, V. Hingorani, N. Jain, A. Karkare, M. Marron, S. Roy. Program synthesis using natural language. In Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 345–356.
[26]. S. Gulwani and M. Marron. Nlyze: Interactive programming by natural language for spreadsheet data analysis and manipulation. In Proceedings of the 2014 ACM SIGMOD international conference on Management of data, 2014, pp. 803–814.
[27]. W. Ling, E. Grefenstette, K. M. Hermann, T. Ko˘ cisk` y, A. Senior, F. Wang, and P. Blunsom. Latent predictor networks for code generation. arXiv preprint arXiv:1603.06744, 2016.
[28]. P. Yin and G. Neubig. A syntactic neural model for general-purpose code generation. arXiv preprint arXiv:1704.01696, 2017.
[29]. X. Gu, H. Zhang, D. Zhang, and S. Kim. Deepam: Migrate apis with multi-modal sequence to sequence learning. arXiv preprint arXiv:1704.07734, 2017.
[30]. X. V. Lin, C. Wang, D. Pang, K. Vu, and M. D. Ernst. Program synthesis from natural language using recurrent neural networks. Technical Report UW-CSE-17-03-01, University of Washington, Department of Computer Science and Engineering, 2017.

Stealth debugging of programs in Qemu emulator with WinDbg debugger

M.A. Abakumov <mikhail.abakumov@ispras.ru>
P.M. Dovgalyuk <dovgaluk@ispras.ru>
Yaroslav-the-Wise Novgorod State University,
41, Great St. Petersburg st., Velikiiy Novgorod, 173003, Russia

Abstract. When programs are analyzed for the presence of vulnerabilities and malicious code, there is a need for a quality isolation of the analysis tools. There are two reasons for this. At first, the program can influence the tool environment. This problem is solved by using the emulator. At second, the tool environment can influence behavior of the analyzed program. So, the programmer will think that the program is harmless, but in fact it is not. This problem is solved by the mechanism of stealth debugging. The WinDbg debugger has the possibility of connecting to a remote debug service (Kdsrv.exe) in the Windows kernel. Therefore, it is possible to connect to the guest system running in the QEMU emulator. Interaction between WinDbg client and server occurs through packets by protocol KDCOM. However, kernel debugging is possible only with the enabled debugging mode in boot settings. And it reveals the debugging process. We developed special module of WinDbg debugger for Qemu emulator. It is an alternative of the remote debugging service in the kernel. Thus, the debugger client tries to connect to the WinDbg server, but module intercepts all packets, generates all the necessary information from the Qemu emulator and sends response to the client. Module completely simulates the behavior of the server, so the client does not notice the spoofing and perfectly interacts with it. At the same time for debugging there is no need to enable debugging mode in the kernel. This leads to stealth debugging. Our module supports all features of WinDbg regarding remote debugging, besides interception of events and exceptions.

Keywords: WinDbg; Qemu; Windows; remote debugging; stealth debugging

DOI: 10.15514/ISPRAS-2018-30(3)-6

For citation: Abakumov M.A., Dovgalyuk P.M. Stealth debugging of programs in Qemu emulator with WinDbg debugger. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018. pp. 87-92. DOI: 10.15514/ISPRAS-2018-30(3)-6

1. Introduction

When performing a dynamic analysis of binary (executable) code, the problem arises of qualitatively isolating the code and the instrumentation on which this

analysis is performed. The need for isolation is dual. On the one hand, it is necessary to limit the impact of the code being studied, since it is able to affect the state of the instrument machine, which is especially important in the study of malicious software. On the other hand, analysis tools can indirectly change the behavior of the program being studied. The most indicative are the situations when errors in working with dynamic memory and race conditions cease to appear in the debugging mode.

The search for undocumented features in a binary code encounters a similar problem. Various techniques and techniques are known [1], with the help of which malware reveals that its execution takes place in a controlled environment, and does not fulfill its objective functions. To find the debugger to be connected, check the int 3 handler and hardware debug registers, evaluate the behavior of certain API functions, and track the progress of the system time.

It is possible to divide potential sources of information, which makes it possible to identify the fact of working in a controlled environment into three disjoint groups. The first is the interaction with external, uncontrolled components, the program being studied, such as remote servers. To the same category, it is necessary to include speed checks. Successful struggle with such sources allows the mechanism of deterministic reproduction [2]. If you write the progress of the system in advance, when debugging and analyzing it during playback, there will be no effect on the guest's state because all time characteristics are fixed during recording. The second group of sources is the discrepancy in the behavior of the equipment [3]. The implementation of virtual equipment in software emulators is not always ideal. Known inaccuracies can be used to determine the emulator in which the program runs. The third group is the analysis tools present in the runtime. This kind of facility occurs even when the debugger is running in conjunction with a virtual machine.

2. Related work

In the Qemu emulator at the moment there is only a module of the GDB debugger, which allows debugging the kernel of the system, but in itself it has relatively small functionality and does not have a GUI. You can use IDA Pro Disassembler [4] ore to connect to the emulator via the GDB interface, but this will not extend the range of the GDB's features, but will only increase the ease of use. In addition, there is a utility called Winbagility [5], which allows the debugger WinDbg to connect to the kernel without debugging mode of the operating system. It is utility for the VirtualBox emulator and is the intermediary between the debugger and the emulator. There is the FDP protocol between Winbagility and the emulator - the introspection API for VirtualBox. It is a minus in this implementation, since the number of provided functions limits the interface.

3. WinDbg

The WinDbg debugger is one of the most advanced debuggers for Windows operating systems. WinDbg is claimed by developers, because it extracts symbolic information from applications and libraries, displays the contents of internal Windows data structures, performs remote debugging of a physical or virtual machine. WinDbg can be used for debugging user applications, device drivers, the operating system itself in kernel mode, to analyze memory dumps in kernel mode created after the so-called Blue Screen of Death, which occurs when an error is issued. It can also be used for debugging custom mode crash dumps. WinDbg supports several debugging modes: debugging the local process, debugging the kernel, and remote debugging.

Target applications can easily detects local debugging process. Remote debugging requires enabled debugging mode in kernel. In this mode kernel uses the debugging server (KdSrv.exe) for interacting with remotely client. But It is also reveals system control (Fig. 1).

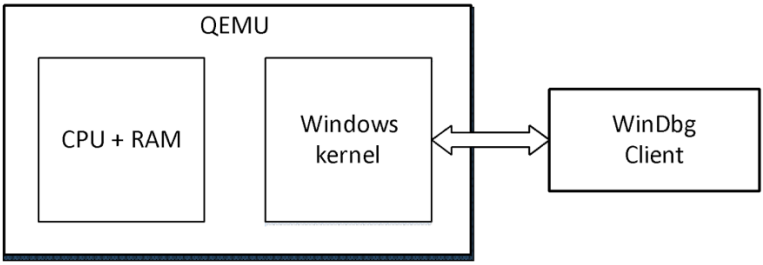


Fig. 1. Direct kernel debugging

4. Stealth debugging

We developed a mechanism for stealth debugging for the QEMU emulator, which allows WinDbg to be remotely connected. The mechanism is an analysis module built into the emulator, and turns out to be an external tool in relation to the guest system. The needs of the KdSrv service in the kernel of the debugging system is not required - the analysis module itself extracts the necessary data from the system and transfers it to the remote client debugger (Fig. 2). The programs running in the guest system cannot track the presence of the connected debugger through functions such as IsDebuggerPresent or through the state of the hardware registers.

One way to remotely kernel debugging using the WinDbg debugger is to debug through the COM port. Interaction between the computers takes place via a private KDCOM protocol, the specification of which has been restored. One of the computers in this case is represented by a virtual machine. The second is an instrumental computer with Windows OS where this machine is started. Running

WinDbg client connects to the emulator via a named pipe, through which the COM-port of the virtual machine is forwarded.

The developed module for the emulator fully implements the KDCOM protocol, within the framework of the restored specification, so the debugger WinDbg interacts with it, as with the debugging module of the Windows kernel, without noticing the substitution. It should be noted that the use of the QEMU emulator as a runtime opens the possibility of debugging during deterministic playback of the virtual machine. The recorded scenarios can be debugged many times in the emulator, which would not be possible if the Windows debug module running inside the guest system were used.

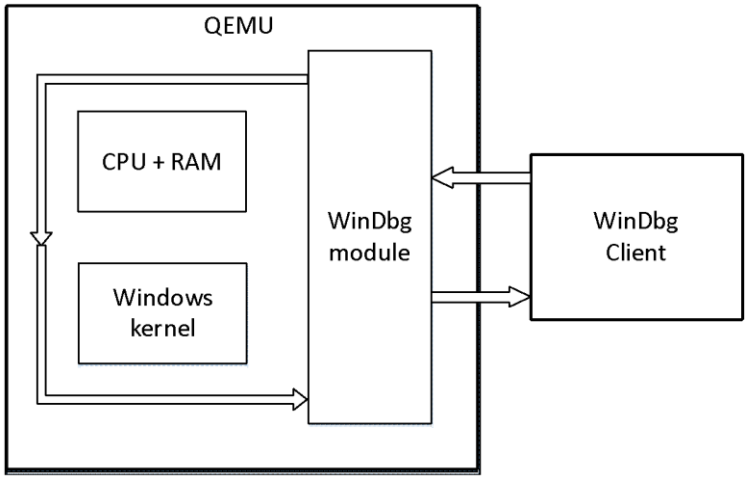


Fig. 2. Kernel debugging through the special module

5. Results and contributons

The developed module supports almost all features of WinDbg regarding remote debugging, besides interception of events and exceptions. It is open source project placed in: github.com/ispras/qemu/tree/windbg. The official community recognized the module as useful. In addition, patches have already been prepared for inclusion in the official repository.

6. Acknowledment

The work was supported by the Ministry of Education and Science of Russia, research project No. 2.6146.2017/8.9.

References

[1]. Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, 2014, pp. 447- 458.

[2]. P. Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 553-556.

[3]. Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In Proceedings of the 3rd USENIX conference on Offensive technologies (WOOT'09). 2009.

[4]. IDA ProDisassembler. Available at: https://www.hex-rays.com/products/ida/index.shtml, accessed 19.06.2018.

[5]. Winbagility. Available at: https://winbagility.github.io/, accessed 19.06.2018.

Скрытая отладка программ отладчиком WinDbg в эмуляторе Qemu

М.А. Абакумов <mikhail.abakumov@ispras.ru>
П.М. Довгалюк <dovgaluk@ispras.ru>
Новгородский государственный университет имени Ярослава Мудрого, 173003, Россия, Великий Новгород, Большая Санкт-Петербургская, д. 41

Аннотация. При анализе программ на наличие уязвимостей и вредоносного кода бывают ситуации, в которых возникает необходимость качественной изоляции инструментов анализа. Этому есть две причины. Во-первых, анализируемая программа может влиять на инструментальную среду. Эта проблема решается использованием эмулятора. Во-вторых, инструменты анализа могут влиять на программу. Так, программист может подумать, что программа безопасна, хотя на самом деле это может быть не так. Эта проблема может быть решена механизмом скрытой отладки. Отладчик WinDbg имеет функцию подключения к удаленному отладочному серверу (Kdsrv.exe), запущенному в ядре Windows. Поэтому есть возможность подключиться к гостевой системе, запущенной в эмуляторе QEMU. Клиент взаимодействует с сервером через пакеты по протоколу KDCOM. Однако отлаживать ядро можно лишь с включенным режимом отладки в настройках запуска, что раскрывает процесс отладки. Мы разработали специальный модуль отладчика WinDbg для QEMU, который является альтернативой удаленному отладочному сервису в ядре. Модуль перехватывает пакеты при взаимодействии клиента отладчика WinDbg с сервером, самостоятельно генерирует всю необходимую отладочную информацию, используя возможности эмулятора Qemu, и отправляет ответ клиенту. Модуль полностью эмулирует поведение отладочного сервера, поэтому клиент на замечает подмены и успешно взаимодействует с ним. При этом отпадает необходимость в отладочном режиме ядра. Тем самым происходит скрытая отладка. При использовании модуля работоспособны все

возможности WinDbg, которые он представляет для удаленной отладки, кроме перехвата событий и исключений.

Ключевые слова: WinDbg; Qemu; Windows; удаленная отладка; скрытая отладка

DOI: 10.15514/ISPRAS-2018-30(3)-6

Для цитирования: Абакумов М.А., Довгалюк П.М. Скрытая отладка программ отладчиком WinDbg в эмуляторе Qemu. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 87-92 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-6

Список литературы

[1]. Timothy Vidas and Nicolas Christin. Evading android runtime analysis via sandbox detection. In Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, 2014, pp. 447- 458.

[2]. P. Dovgalyuk. Deterministic Replay of System's Execution with Multi-target QEMU Simulator for Dynamic Analysis and Reverse Debugging. In Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 553-556.

[3]. Roberto Paleari, Lorenzo Martignoni, Giampaolo Fresi Roglia, and Danilo Bruschi. A fistful of red-pills: how to automatically generate procedures to detect CPU emulators. In Proceedings of the 3rd USENIX conference on Offensive technologies (WOOT'09). 2009.

[4]. IDA Pro Disassembler. Режим доступа: https://www.hex-rays.com/products/ida/index.shtml, дата обращения 19.06.2018.

[5]. Winbagility. Режим доступа: https://winbagility.github.io/, дата обращения 19.06.2018.

Configurable system call tracer in QEMU emulator

A.V. Ivanov <alexey.ivanov@ispras.ru>
P.M. Dovgaluk <pavel.dovgaluk@ispras.ru>
V.A. Makarov <vladimir.makarov@ispras.ru>
Yaroslav-the-Wise Novgorod State University,
41, Great St. Petersburg st., Velikiiy Novgorod, 173003, Russia

Abstract. Sometimes programmers face the task of analyzing the work of a compiled program. To do this, there are many different tools for debugging and tracing written programs. One of these tools is the analysis of the application through system calls. With a detailed study of the mechanism of system calls, you can find a lot of nuances that you have to deal with when developing a program analyzer using system calls. This paper discusses the implementation of a tracer that allows you to analyze programs based on system calls. In addition, the paper describes the problems that I had to face in its design and development. Now there are a lot of different operating systems and for each operating system must be developed its own approach to implementing the debugger. The same problem arises with the architecture of the processor, under which the operating system is running. For each architecture, the analyzer must change its behavior and adjust to it. As a solution to this problem, the paper proposes to describe the operating system model, which we analyze. The model description is a configuration file that can be changed depending on the needs of the operating systems. When a system call is detected the plugin collects the information downloaded from the configuration file. In a configuration file, arguments are expressions, so we need to implement a parser that needs to recognize input expressions and calculate their values. After calculating the values of all expressions, the tracer formalizes the collected data and outputs it to the log file.

Keywords: QEMU; configurable system calls; debugging; plugin; system calls; tracing.

DOI: 10.15514/ISPRAS-2018-30(3)-7

For citation: Ivanov A.V., Dovgaluk P.M., Makarov V.A. Configurable system call tracer in QEMU emulator. Trudy ISP RAN/Proc. ISP RAS, Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 93-98. DOI: 10.15514/ISPRAS-2018-30(3)-7

1. Introduction

Sometimes programmers face the task of analyzing the work of a compiled program to find its flaws, defects, and even search for malicious code in it. To analyze the work of such applications, we have to study their binary code or try to decompile the code, which is a laborious task. In order to simplify the analysis of applications,

we can use the system calls of this application. System calls provide an essential interface between a program and the operating system. It is possible to track which system calls the application makes, and draw conclusions about the behavior of the program. This method allows us to debug the application without delving into the level of instructions and architecture features, thereby reducing the time required to find the problem.

Debugging applications using system tracing can be done inside the operating system, but still a number of problems arise:

- strong dependence of the debugger on the operating system;
- impossibility to run several debuggers at the same time;
- inaccessibility to the privileged execution;
- necessity to secure the operating system when analyzing programs that have harmful effects.

To solve these problems, we can use the virtual machine tools. In this way, we can debug applications in a wide range of different operating systems running under different processor architectures.

2. Approach and uniqueness

To date, several debuggers allow us to trace an application using system calls. All these debuggers have a drawback - they do not provide enough portability of the debugger within different operating systems and processor architectures. We offer a new approach to implementing the debugger through system calls, by loading all the information necessary for tracing from the configuration file. The configuration files will allow us to easily configure and change the parameters needed for debugging, and to simplify the addition of support for new operating systems and architectures without recompiling the program and learning the debugger code.

It was decided to implement the debugger under the virtual machine QEMU [1], using the plugin mechanism. QEMU is an open source virtual machine that emulates the hardware of various platforms. This virtual machine supports the emulation of a large number of processor architectures such as x86, PowerPC, ARM, MIPS, SPARC, m68k. In addition, this simulator supports the launch of a large number of different operating systems.

Now, there is a plugin mechanism for QEMU implemented by ISP RAS [2], which allows us to connect developed plugins to a virtual machine during its both startup and operation. The implementation of the plugin mechanism enables each additional translation of the instruction to be substituted by an additional code for execution, when this instruction is called. This mechanism is suitable for debugging through system calls, so it was decided to use it.

In addition, various mechanisms of the system call play an important role. The classical way of implementation is the use of interrupts. With the help of interrupts, control is transferred to the kernel of the operating system, with the application

having to enter the number of the system call and the necessary arguments into the corresponding registers of the processor.

For many RISC processors, this method is the only one; however, the CISC architecture has additional methods. The two mechanisms developed by AMD and Intel are independent of each other, but, in fact, perform the same functions. These are SYSCALL / SYSRET or SYSENTER / SYSEXIT statements. They allow us to transfer control to the operating system without interrupts.

Each operating system supports values returned from the system call, which are passed as reference types when the system call handler is called. During the execution of the system call, the service procedure records the required values if necessary by the available links, after which the system call is exited.

One of the main tasks that we had to face was the task of supporting the plugin of different operating systems and processor architectures. The solution to this problem was the interface with the configuration file. The configuration file makes the debugger more flexible and customizable. With its help, we can disconnect a certain mechanism of system calls from the trace or disable unnecessary system calls. In addition, such a mechanism makes it easier to add support for new operating systems and processor architectures.

To implement the interface with the configuration file, it was necessary to study a wide range of different operating systems and processor architectures. After gathering the necessary information, we can determine the information necessary for debugging: what type of system call is supported by SYSCALL / SYSRET or SYSENTER / SYSEXIT and their opcodes; location of system call arguments; a list of system calls, with the name of each system call, its code, and the list of arguments. Thus, by developing an interface for debugger and configuration file interfacing, we can add support for operating systems without going into the debugger code.

When implementing the debugger interaction interface with the configuration file, it became necessary to recognize the various expressions read from the file. For this task, we used the generator of the bison parser and developed the corresponding grammar [3].

3. Background and related work

Now, there are several debuggers to solve existing problems. Nitro [4] allows us to trace system calls, but it works only under Intel x86 architecture. Another debugger – Panda [5], can also trace system calls, supporting such operating systems as Linux, Windows 7, Windows XP and two architectures of the i386 processor and ARM. The description of all system calls is found in the code of this debugger, because of which this approach makes it difficult to add support for new operating systems and processor architectures, and worsens the flexibility in configuring the plugin, since the system debugger settings mechanism is not provided.

4. Conclusion and discussion

Based on the results of the work done, the plugin was developed in the QEMU virtual machine, with which we can trace and debug an application using system calls. As input to the plugin, the configuration file corresponding to the operating system running in the QEMU virtual machine and corresponding to the selected processor architecture is used.

The structure of the configuration file consists of 4 parts. The first part provides information about the operating system, its name and bit capacity. The second part is responsible for the supported mechanisms of system calls. The next part contains the location of the system call arguments. The last part includes a list of all available system calls and service information about the arguments of the system calls.

Because of the plugin's work, a log file containing all the system calls that the plugin has intercepted is created. Each system call displays detailed information: the name and value of each system call argument, the number of the thread of execution from which this system call was made and the value that returned the system call after execution. Fig. 1 presents a small fragment of the output file that was created by the implementation of the plugin launched in the windows XP operating system and the i386 processor architecture.

```
0x3e84000 entr:  0x114: NtWriteRequestData
0x3e84000 exit:  0x114: NtWriteRequestData
    return: 0x0
0x3e84000 entr:  0xc4: NtReplyWaitReceivePortEx
0x3e84000 entr:  0x112: NtWriteFile
    arg 0: 0x2a4      ( HANDLE FileHandle )
    arg 1: 0x0        ( HANDLE Event )
    arg 2: 0x0        ( PIO_APC_ROUTINE ApcRoutine )
    arg 3: 0x0        ( PVOID ApcContext )
    arg 4: 0x8ff6d8   ( PIO_STATUS_BLOCK IoStatusBlock )
    arg 5: 0x9059f8   ( PVOID Buffer )
    arg 6: 0xbc       ( ULONG Length )
    arg 7: 0x8ff6e0   ( PLARGE_INTEGER ByteOffset )
    arg 8: 0x0        ( PULONG Key )
0x3e84000 exit:  0x112: NtWriteFile
    return: 0x0
0x3e84000 entr:  0x74: NtOpenFile
    arg 0: 0x8ff6c4   ( PHANDLE FileHandle )
    arg 1: 0x100100   ( ACCESS_MASK DesiredAccess )
    arg 2: 0x8ff680   ( POBJECT_ATTRIBUTES ObjectAttributes )
    arg 3: 0x8ff6a4   ( PIO_STATUS_BLOCK IoStatusBlock )
    arg 4: 0x7        ( ULONG ShareAccess )
    arg 5: 0x204020   ( ULONG OpenOptions )
0x3e84000 exit:  0x74: NtOpenFile
    return: 0x0
0x3e84000 entr:  0xe0: NtSetInformationFile
    arg 0: 0x31c      ( HANDLE FileHandle )
    arg 1: 0x8ff6a4   ( PIO_STATUS_BLOCK IoStatusBlock )
    arg 2: 0x8ff658   ( PVOID FileInformation )
    arg 3: 0x28       ( ULONG Length )
    arg 4: 0x4        ( FILE_INFORMATION_CLASS FileInformationClass )
0x3e84000 exit:  0xe0: NtSetInformationFile
    return: 0x0
0x3e84000 entr:  0x19: NtClose
    arg 0: 0x31c      ( HANDLE Handle )
0x3e84000 exit:  0x19: NtClose
    return: 0x0
```

Fig. 1. Part of the output file of the plugin

Upon the information gathered in the log file, we can analyze the operation of the debugged application running inside the virtual machine. The operating system load

time when using the developed plugin is increased 20% slowdown compared to the time of the operating system loading without this plugin.

Acknowledgments

The work was supported by the Russian Foundation of Basic Research (research grant 18-07-00900 A)

References

[1]. F. Bellard. QEMU, a fast and portable dynamic translator. In Proceedings of the Annual Conference on USENIX Annual Technical Conference, 2005.
[2]. Vasiliev I.A., Fursova N.I., Dovgaluk P.M., Klimushenkova M.A., Makarov V.A. Modules for instrumenting the executable code in QEMU. Problemy informacionnoj bezopasnosti. Komp'juternye sistemy [Journal of Information Security Problems. Computer Systems], no. 4, 2015, pp. 195-203 (in Russian).
[3]. GNU Bison [HTML] (https://www.gnu.org/software/bison/)
[4]. Nitro [HTML] (http://nitro.pfoh.net/index.html)
[5]. Panda. Plugin: syscalls2. [HTML] (https://github.com/panda-re/panda/blob/master/panda/plugins/syscalls2/USAGE.md)

Конфигурируемый трассировщик системных вызовов в эмуляторе QEMU

А.В. Иванов <alexey.ivanov@ispras.ru>
П.М. Довгалюк <pavel.dovgaluk@ispras.ru>
В.А. Макаров <vladimir.makarov@ispras.ru>
Новгородский государственный институт имени Ярослава Мудрого,
173003, Россия, г. Великий Новгород, ул. Б. Санкт-Петербургская, д. 41

Аннотация. Разработчики программ часто сталкиваются с проблемой анализа работы различных приложений. Для этого существует большое множество различных средств отладки, отслеживания, трассировки написанных программ. Одним из таких средств является анализ работы приложения через системные вызовы. При детальном изучении механизма системных вызовов, можно обнаружить большое количество нюансов, с которыми приходится столкнуться при разработке анализатора программ с использованием системных вызовов. В статье рассматривается реализация трассировщика, который позволяет анализировать программы на основе системных вызовов, и проблемы, с которыми пришлось столкнуться при его проектировании и разработке. На данный момент существует большое количество различных операционных систем и для каждой операционной системы должен быть разработан свой подход в реализации отладчика. Такая же проблема возникает и с архитектурой процессора, под которой запущена операционная система. Для каждой архитектуры, анализатор должен менять своё поведение и подстраиваться под неё. В качестве решения данной проблемы, в статье предлагается описать модель операционной системы, которую мы анализируем. Описание модели представляет собой конфигурационный файл, который может быть изменён в зависимости от потребностей

операционных систем. При обнаружении системного вызова, в его обработчик передаются аргументы и вся сопутствующая информация, загруженная из конфигурационного файла. Изначально, в конфигурационном файле, все аргументы представляют собой выражения, поэтому возникает необходимость также реализовать синтаксический анализатор, которому необходимо распознать входные выражения и посчитать их значения. После просчёта значений всех выражений, трассировщик формализует собранные данные и выводит их в лог файл.

Ключевые слова: QEMU; конфигурируемые системные вызовы; настраиваемые системные вызовы; отладка; отладчик; плагин; системные вызовы; трассировка; трассировщик.

DOI: 10.15514/ISPRAS-2018-30(3)-7

Для цитирования: Иванов А.В., Довгалюк П.М., Макаров В.А. Конфигурируемый трассировщик системных вызовов в эмуляторе QEMU. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 93-98 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-7

Список литературы

[1]. F. Bellard. Qemu, a fast and portable dynamic translator. In Proceedings of the Annual Conference on USENIX Annual Technical Conference, 2005.
[2]. Васильев И.А., Фурсова Н.И., Довгалюк П.М., Климушенкова М.А., Макаров В.А. Модули для инструментирования исполняемого кода в симуляторе QEMU. Проблемы информационной безопасности. Компьютерные системы, no, 4, 2015г., стр. 195-203
[3]. GNU Bison [HTML] (https://www.gnu.org/software/bison/)
[4]. Nitro. [HTML] (http://nitro.pfoh.net/index.html)
[5]. Panda. Plugin: syscalls2. [HTML] (https://github.com/panda-re/panda/blob/master/panda/plugins/syscalls2/USAGE.md)

Анализ методов оценки надежности оборудования и систем. Практика применения методов

[1,2]Е.М. Лаврищева <lavr@ispras.ru>
[1,2,3] Н.В. Пакулин<npak@ispras.ru>
[1]А.Г. Рыжов <ryzhov@ispras.ru>
[1,3]С.В. Зеленов <zelenov@ispras.ru>
[1]Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25
[2] Московский физико-технический институт,
141701, Россия, Московская обл., г. Долгопрудный, Институтский пер., д. 9
[3]Национальный исследовательский университет «Высшая школа экономики»
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Проводится анализ моделей и методов оценки надежности технических и программных средств. Определяются основные понятия методов надежности и безопасности таких систем и ситуаций, приводящих к ошибкам, дефектам и отказам. Дано определение надежности и безопасности технических систем и программного обеспечения (ПО) систем. Приведена классификация моделей надежности: прогнозирующего, измерительного и оценочного типов. Описаны оценочные модели, которые применяются на практике. Определен стандарт жизненного цикла ПО (ISO 15288:2002), ориентированный на разработку и контроль компонентов систем на ошибки, начиная с требований к системе. Представлены результаты применения моделей надежности (Мусы, Гоэла-Окомото и др.) к малым, средним и большим проектам и дана сравнительная их оценка. Описан технологический модуль (ТМ) оценки надежности сложных комплексов программ ВПК (1989). Показана модель качества стандарта ISO 9126 (1-4):2002-2004 с показателями функциональность, надежность, эффективность и др., которые используются при определении зрелости и сертификата качества продукта.

Ключевые слова: надежность; ошибка; дефект; отказ; плотность дефектов; случайный процесс; безопасность; гарантоспособность; восстанавливаемость; отказоустойчивость; завершенность; оценка надежности; сертификат качества

DOI: 10.15514/ISPRAS-2018-30(3)-8

Для цитирования: Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 99-120. DOI: 10.15514/ISPRAS-2018-30(3)-8

1. Введение

Рассматривается теория надежности и безопасности технических и программных систем [1-10]. Теория надежности техники возникла в рамках теории массового обслуживания систем и повлияла на становление и развитие теории надежности компьютерных систем и программного обеспечения (ПО) систем.

Теоретики, изучая природу ошибок в системах, разработали более 100 математических моделей надежности, основанных на учете различных ситуаций, возникающих в технических и программных системах на первых и последующих поколениях ЭВМ. Методы надежности обеспечивают повышение надежности систем путем исправления разного рода обнаруженных ошибок в процессе разработки и их эксплуатации.

Надежность систем сформировалась как самостоятельная теоретическая и прикладная наука, способствующая определению качественных показателей систем (функциональность, точность, отказоустойчивость, гарантоспособность, завершенность и др.).

Методы оценки надежности систем позволяют прогнозировать, измерять и оценивать качество продукта с учетом возникающих ошибок, количества и интенсивности отказов, а также процессов разработки отдельных компонентов систем в жизненном цикле (ЖЦ).

В работе рассматриваются все аспекты обеспечения надежности, безопасности и качества технических и программных систем [1-11].

2. Методы надежности оборудования и систем

Методы оценки надежности технических систем (аппаратуры, устройства, оборудование и др.) были разработаны значительно ранее компьютерных систем и основывались на вероятностных Марковских процессах с множеством числа состояний по теории массового обслуживания [12-16]. Эти методы обеспечивали проверку надежности функционирования техники, приборов и устройств в различных областях (машиностроение, энергетика, космос, медицина и др.). На их работоспособность влияли неисправности и разные недоработки в конструкции, приводящие к разрушительным последствиям и к ущербу системы в целом.

На оценку надежности компьютерных систем и ПО существенным образом влияют следующие особенности.

1. Большое количество кода в ПО, зачастую превышающее емкость физических элементов ЭВМ, и способов взаимодействия отдельных элементов ПО между собой;
2. Нематериальный характер элементов ПО, которые не деградируют, но стареют во времени, и в их процессах, программах и конструкциях могут случаться разного рода непредвиденные ситуации;

3. Возникновение ошибочных ситуаций, разных дефектов и отказов как в задании формальной спецификации отдельных элементов, так и в их выходном коде;
4. Элементы ПО трудно поддаются визуализации, обнаружению и коррекции найденных ошибок, поэтому измерение надежности ПО требует анализа и проверки данных об ошибках в большей степени, чем для аппаратуры;
5. Системы ПО могут изменяться при функционировании и выходить из рабочего состояния от разных ситуаций внешней среды (вирусов, атак и др.), которые не предусмотрены в соответствующих моделях надёжности и обеспечиваются методами безопасности информации и систем.

Надежность технических систем зависит от двух факторов:

- качества отдельных технических конструктивных элементов системы;
- отсутствия неисправностей в конструктивных элементах и их способность работать надежно и качественно.

Надежность программных систем зависит от этих же факторов и от случайных изменений данных и маршрутов исполнения программ, которые могут привести к неверным результатам или отказам.

Между надежностью аппаратуры и ПО систем имеется сходство, состоящее в возникновении случайных явлений в процессах и системах, которые должны анализироваться методами теории вероятности, надежности и безопасности.

2.1. Определение термина надежности и безопасности систем

Под ***надежностью систем*** понимается способность системы сохранять свои свойства (безотказность, восстанавливаемость, защищенность и др.) на заданном уровне в течение фиксированного промежутка времени при определенных условиях эксплуатации.

Термин ***надежность*** (reliability) обозначает способность системы обладать свойствами, обеспечивающими выполнение функций системы в соответствии с заданными требованиями.

Термин ***гарантоспособность*** (dependability) означает способность системы гарантировать выполнение услуг, для которых она предназначена, и состоит в:

- безотказности выполнения;
- готовности к работе;
- достоверности результатов;
- приспособленности к обслуживанию или ремонту (maintainability);
- информационной безопасности (security);
- конфиденциальности (confidentiality), секретности и целостности информации (integrity);

- безопасности (safety) работы системы без катастрофических последствий;
- эксплуатационной завершенности ПО и способности к восстановлению работоспособности системы.

Отказоустойчивость (fault-tolerance) обозначает способность системы автоматически за ограниченное время прогнозировать, предупреждать и восстанавливать функциональность системы после отказов с помощью механизмов поддержки всех составляющих гарантоспособности. Системы или процессы, которые обладают таким комплексным свойством, называют гарантоспособными. Им присущи традиционные надежностные свойства (безотказность, готовность, безопасность, целостность, конфиденциальность, восстанавливаемость и др.).

Вопросы разработки и использования гарантии качества систем обсуждаются более 25 лет на международных форумах и конференциях (Conference on Dependable Systems and Networks (DSN), European Dependable Computing Conference, (EDCC), International Conference on Computer Safety, Reliability and Security (SAFECOM), Probabilistic Safety Assessment and Management Conference (PSAM), Dependable Systems, Services and Technologies (DESSERT), Conference on Dependability of Complex Systems (DepCoS) и др.).

С 2004 году ассоциация IEEE издает журнал Dependable and Security Computing. В нем обсуждаются бизнес-критические приложения, электронная коммерция, банковские технологии и др. [16, 17].

С точки зрения гарантоспособности надежность является целевой функцией реализации системы. К ней предъявляются высокие требования (недопустимость ошибок, отказов, дефектов и других аварийных ситуаций). Надежность систем зависит от числа оставшихся и не устраненных ошибок в отдельных программах и компонентах системы.

Чем интенсивнее проводится эксплуатация системы, тем интенсивнее выявляются ошибки, быстрее растет надежность системы и соответственно ее качество. Надежность, по существу, является функцией от ошибок, оставшихся в системе после ввода ее в эксплуатацию. Системы без ошибок считаются абсолютно надежными. Для оценки надежности систем используются собранные статистические данные – время безотказной работы, дефекты и частота (интенсивность) отказов.

Исследование надежности систем проводится с помощью методов теории вероятностей, математической статистики, теории массового обслуживания и математических методов надежности и безопасности. Главным источником информации для оценки надежности являются процессы тестирования, эксплуатации и испытания системы и данные, полученные при разработке систем в соответствии со стандартами жизненного цикла (ЖЦ) (ISO/IEC 15846-1998, 15939:2002) системной инженерии [15-23].

2.2. Базовые понятия моделей надежности и безопасности

К базовым понятиям, которые используются в моделях надежности систем, относятся следующие [1-5].

- *Отказ* (failure) – это переход системы из рабочего состояния в нерабочее.
- *Дефект* (fault) – это последствие выполнения элемента программы, приводящее к некоторому непредвиденному событию (неверной интерпретации компьютером); невыявленные дефекты – источник потенциальных ошибок и отказов системы.
- *Ошибка* (error) может быть следствием недостатка в спецификациях любой из программ или при принятии неверных действий в процессе испытания системы.
- *Интенсивность отказов* – это частота появления отказов или дефектов в системе при ее тестировании, эксплуатации и сопровождении системы.

Одним из подходов к исследованию надежности на основе отказов систем является классическая теория вероятностей, согласно которой отказы в системе считаются случайными и зависят от ошибок, внесенных при разработке системы. Все модели оценки надежности базируются на статистике отказов и интенсивности выявленных отказов в процессе верификации, тестирования и испытания системы для обеспечения ее работоспособности и гарантоспособности [16, 17].

Модели надежности основываются на нахождении случайной величины в системе, числом и интенсивностью возникновения отказов в системе. Для случайной величины вычисляется математическое ожидание и дисперсия (среднее отклонение). Если случайная величина дискретна, т.е. принимает конечное число значений x_1, x_2, …, x_n , то ее распределение описывается вероятностью $P(\xi = x_i)$, и в общем случае $F(x) = P(\xi < x_i)$ является функцией распределения случайной величины.

Случайный процесс с непрерывным временем, который описывается однородными событиями, называется пуассоновским процессом.

Если случайные величины, полученные на всем ЖЦ системы, распределены по показательному, эрланговскому или гиперэрланговскому законам, то поведение системы описывается Марковским процессом без непрерывных компонентов.

Надежность по существу очень близка задачам безопасности. При разработке систем научными и некоммерческими институтами их трудно оценить на надежность и безопасность из-за того, что они делаются, как правило, не по стандартам. В то время как системы управления авиацией, атомной энергетикой и оборонной промышленностью разрабатываются по стандартам. В них надежность и безопасность определяют работоспособность системы в соответствии с требованиями и с минимум отказов и дефектов.

В характеристиках безопасности учитываются только те отказы, которые могут привести к катастрофическим последствиям и ущербам (например, пожар, взрыв, разрушение здания и др.). Оценка безопасности системы базируется на надежности функционирования ПО и БД. Оценка надежности зависит от метрик стандарта качества (внешние, внутренние, эксплуатационные). Они сравниваются с требованиями заказчика на систему и используются при сертификации продукта. Для оценки надежности и функциональной безопасности используются стандарты ISO/IEC 12207 для ПО и ISO 15288 -2006 систем.

На работоспособность системы влияют отказы, дефекты и ошибки проектирования, которые приводят к длительности восстановления и к необходимости устранять в программе ошибки средствами программной и информационной избыточности. Согласно стандарта ISO 9126 (1-4) определяются характеристики надежности с учетом обнаруженных дефектов и ошибок при функционировании гарантоспособного ПО систем.

Степень работоспособности/гарантоспособности зависит от соответствия характеристик требований, заданных в проекте, выявленным ошибкам и отказам в ПО и возможным неисправностям в конструктивных элементах систем.

3. Классификация моделей надежности ПО

Большинство моделей надежности исходят из предположения, что найденные ошибки и дефекты устраняются немедленно или определяются временем их устранения и новые дефекты не вносятся. В результате количество дефектов в системе уменьшается, а надежность возрастает, такие модели получили название моделей роста надежности. Shick G. [6, 16, 17] предложил следующую классификацию моделей надежности.

Прогнозирующие модели надежности основаны на измерении технических характеристик создаваемой программы: длина, сложность, число циклов и степень их вложенности, количество ошибок на страницу операторов программы и др.

Модель Мотли–Брукса основывается на длине и сложности структуры программы (количество ветвей и циклов, вложенность циклов), количестве и типах переменных, а также интерфейсов.

Модель Холстеда дает прогнозирование количества ошибок в программе в зависимости от ее объема и таких данных, как число операций (n_1) и операндов (n_2), а также общее число обращений к ним ($N_1 + N_2$).

Измерительные модели предназначены для измерения надежности ПО, работающего с заданной внешней средой и следующими ограничениями:

- ПО не модифицируется во время периода измерений свойств надежности;
- обнаруженные ошибки не исправляются;

- измерение надежности проводится для зафиксированной конфигурации ПО.

Примером таких моделей является модель Нельсона, Рамамурти–Бастани и др.[3].

Модель Нельсона основывается на выполнении k прогонов программы при тестировании и позволяет определить надежность по формуле:

$R(k) = \exp[-\sum t_j \lambda(t_j)]$,

где t_j – время выполнения j-го прогона, $\lambda(t_j) = -\ln(1-q_j)/t_j$ и при $q_i \leq 1$ интерпретируется как функция интенсивности отказов.

Оценочные модели основываются на серии тестовых прогонов и проводятся на этапах тестирования системы. В тестовой среде определяется вероятность отказа программы при ее тестировании или выполнении. Эти типы моделей могут применяться на этапах ЖЦ и могут быть следующих видов [7-11].

Модели без подсчета ошибок основаны на измерении интервала времени между отказами и позволяют спрогнозировать количество ошибок, оставшихся в программе. К этим моделям относятся модели Джелински и Моранды, Шика Вулвертона, и Литвуда–Вералла.

Модели с подсчетом отказов базируются на количестве ошибок, обнаруженных на заданных интервалах времени. К этому классу моделей относятся модели Шика–Вулвертона, Шумана, Пуассоновская модель и др.

Модели с подсевом ошибок основаны на количестве устраненных ошибок и подсеве, внесенном рограмму искусственных ошибок, тип и количество которых заранее известны. При внесении изменений в программу проводится повторное тестирование и оценка надежности. Этот подход базируется на тестировании и редко используется из-за дополнительного объема работ для покрытия тестами компонентов системы.

Модели с выбором области входных значений основываются на генерации множества тестовых выборок из входного набора. К этому типу моделей относится модель Нельсона и др. На процессах выявления отказов и их интенсивности используются также:

1) модели, рассматривающие интенсивность отказов, как Марковский и пуассоновский процесс;
2) модели роста надежности.

Четкой границы между этими моделями провести нельзя, однако по фактору интенсивности отказов и моделей поведения их можно разделить на экспоненциальные, логарифмические, геометрические, байесовские и др.

Для практической оценки надежности более всего представляет интерес оценочная модель Мусы, Мусы-Окомото и др. Рассмотрим их.

1. Оценочная модель Мусы [8] основана на следующих положениях:

- тесты адекватно представляют среду функционирования;
- происходящие отказы учитываются (оценивается их количество);

- интервалы между отказами независимы;
- время между отказами распределено по экспоненциальному закону;
- интенсивность отказов пропорциональна числу ошибок;
- скорость исправления ошибок (относительно времени функционирования)
- пропорциональна интенсивности их появления.

Эта модель учитывает интервалы между отказами, которые распределяется по экспоненциальному закону, а интенсивность отказов пропорциональна числу обнаруженных ошибок.

Исходя из этой модели, можно установить зависимость:

1) среднего числа отказов от времени функционирования τ (рис.1), которое задается в виде:

$$m=M_0\,[1-exp\,(\frac{c\tau}{M_0T_0})],$$

где M_0 – общее число ошибок; T_0 – начальная наработка на отказ; с – коэффициент времени испытаний; τ – время функционирования.

2) средней наработки на отказ Т от времени функционирования τ (рис.2):

$T = T_0 \exp(\frac{c\tau}{M_0T_0})$, где M_0, T_0, с – зависят от наработки на отказ.

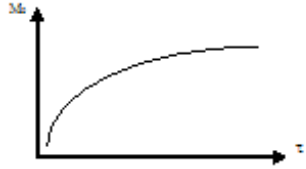


Рис. 1. Зависимость числа отказов от времени τ
Fig. 1. The dependence of the number of failures on time of τ

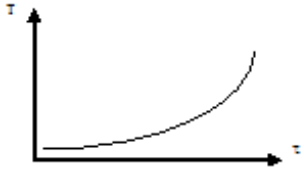


Рис. 2. Зависимость средней наработки на отказ от времени τ
Fig. 2. The dependence mean time to failure rate from the time τ

График этой зависимости представлен областью 1 (рис. 3), для которой M_1 = 1, 2, ... – номера наблюдений, а $\tau_1, \tau_2 \ldots \tau_{M1}$ – время между отказами. Область 2 (рис.3) соответствует достижению средней наработки T_p на отказ за время Δτ.

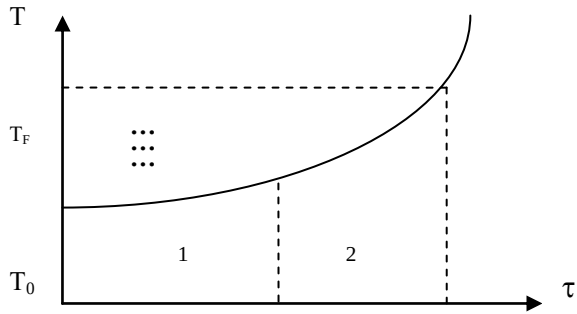


Рис. 3. Области отказов и наработки на отказ от времени τ
Pic. 3. The field of failures, and mean time to failure rate from τ

По собранным данным об ошибках оцениваются параметры T0 и M0, с помощью которых определяются дополнительное число ошибок по формуле:

$$\Delta m = M_0 T_0 [\frac{1}{T} - \frac{1}{T_0}].$$

2. Модель Мусы–Окумото (логарифмическая) допускает, что некоторые ошибки имеют большую вероятность проявления в виде отказов, снижают интенсивность отказов с каждой устраненной ошибкой и дают экспоненциальное распределение. Мат.ожидание найденных ошибок $m(t)$ имеет вид:

$m(t) = (1/\theta) \ln(\lambda_0 \theta t + 1)$,

где λ_0 – исходная интенсивность отказов, θ – скорость спада интенсивности отказов с каждым устраненным дефектом, а функция интенсивности отказов $\lambda(t)$ имеет вид:

$\lambda(t) = \lambda_0 / (\lambda_0 \theta t + 1)$.

3. Модель Гоело–Окумото (экспоненциального роста) описывает обнаружение ошибок с помощью неоднородного пуассоновского процесса. В ней интенсивность отказов зависит от времени, а количество выявленных ошибок при тестировании трактуется как случайная величина.

Исходные данные m, X_i и T аналогичны данным предыдущих моделей. Функция среднего числа отказов, обнаруженных к моменту t, имеет вид:

$m(t) = N(1 - e^{-bt})$,

где b – интенсивность обнаружения отказов; $q(t)=b$ - показатель роста надежности.

Функция интенсивности $\lambda(t)$ зависит от времени работы системы до отказа:

$\lambda(t) = Nbe^{-bt}$, $t \geq 0$,

где N и b решаются с помощью уравнения*:*

$- m/N - 1 + \exp(-bT) = 0$

4. S-образная модель. Функция интенсивности $\lambda(t)$ выявления ошибок в зависимости от времени работы имеет вид:

$\lambda(t) = a\beta^2 t \exp(-\beta t)$,

где a – общее количество дефектов, обнаруженных от начала и до конца тестирования;

β – скорость изменения функции интенсивности выявления отказов.

Введение в формулу параметра в степени 1 модели Мусы и Гоэла–Окомото дает изменение формы кривой так, что она сначала растет, а потом спадает. Практика применения этих моделей в автоматизированных системах привела к уточнению функции интенсивности при введении дополнительного параметра n:

$\lambda(t) = a\beta^{n+1} t^n \exp(-\beta t)$,

где n отражает сложность и размер проекта некоторой системы. Это позволяет более точно определить форму кривой интенсивности с учетом получаемых практических результатов.

Процесс оценки надежности включает:

- протоколирование отказов в ходе функционирования системы, измерение надежности по отказам и использование результатов измерений для определения потерь надежности в период времени эксплуатации;
- анализ частоты и серьезности отказов при определении порядка устранения соответствующих ошибок;
- оценка влияния времени функционирования системы на надежность с помощью инструментов измерения надежности.

3.1. Оценка надежности систем реального времени

Некоторые типы систем реального времени с обеспечением безопасности требуют высокой надежности (недопустимость ошибок, точность, достоверность и др.), которая в значительной степени зависит от количества оставшихся и не устраненных ошибок в процессе ее разработки на этапах ЖЦ.

В ходе эксплуатации системы также могут обнаруживаться и устраняться ошибки. Если при их исправлении не вносятся новые ошибки или их меньше, чем устраняется, то в ходе эксплуатации надежность системы непрерывно растет. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки и быстрее растет надежность.

На надежность ПО влияют, с одной стороны, угрозы, приводящие к неблагоприятным последствиям, риск нарушения безопасности системы, с другой стороны, способность совокупности компонентов системы сохранять устойчивость в процессе ее эксплуатации. Риск уменьшает свойства

надежности, особенно если обнаруженные ошибки могут быть результатом проявления угрозы извне [16].

Методы и модели надежности постоянно развиваются и уточняются, поскольку надежность является одной из ключевых проблем измерения качества современных распределенных по Интернет систем.

Появилось новое направление – *инженерия надежности ПО* (Software reliability engineering – SRE), которое ориентировано на количественное изучение операционного поведения компонентов системы по отношению к пользователю, ожидающему получить надежную работу системы [16-20]. Надежность обеспечивается путем:

1) измерения надежности, т.е. проведения количественной ее оценки методами предсказаний, собранными данными о поведении системы в процессе тестирования и эксплуатации системы;
2) оценки стратегии и применения метрик для готовых компонентов, созданных в процессе разработки компонентов системы в заданной среде и стандартов на измерение надежности системы;
3) применения современных методов инспектирования, верификации, валидации и тестирования в процессе разработки отдельных компонентов и системы в целом.

3.2. Обеспечение надежности на этапах ЖЦ

Для получения высокой надежности системы требуется наблюдать за достижением показателей надежности и качества на всех этапах ЖЦ согласно рекомендациям стандарта ISO/IEC 12207: ЖЦ [16]. К основным процессам стандарта ЖЦ относятся:

- спецификация требований,
- проектирование,
- реализация,
- тестирование,
- испытание,
- сопровождение.

На этапе спецификации требований определяются задачи и внешние спецификации основных (целевых) требований к системе с заданием метрик для оценки надежности, в терминах интенсивности отказов или вероятности безотказного его функционирования. Разработчики системы формируют:

- приоритеты функций системы по критерию важности их реализации;
- параметры среды и интенсивности использования функций и их отказов;
- входные и выходные данные для каждой функции системы;
- категорий отказов и их интенсивности при выполнении функций в единицу календарного времени.

На этапе проектирования определяются:

- размеры информационной и алгоритмической сложности всех типов проектируемых компонентов;
- категории дефектов, свойственные всем типам компонентов системы;
- стратегии функционального тестирования компонентов по принципу «черного ящика» с помощью тестов для выявления ошибок в данных.

Для обеспечения надежности продукта проводится:

- анализ вариантов архитектуры системы на соответствие требованиям к надежности;
- анализ рисков, режимов отказов, деревьев ошибок для критических компонентов с целью обеспечения отказоустойчивости и восстанавливаемости системы;
- прогнозирование показателей размера системы, чувствительности к ошибкам, степени тестируемости, оценки риска и сложности системы.

На этапе реализации и тестирования системы проектные спецификации переводятся в коды и подготавливаются наборы тестов для автономного и комплексного их тестирования. При проведении автономного тестирования обеспечение надежности состоит в предупреждении появления дефектов в компонентах и создание эффективных методов защиты от них. Все последующие этапы разработки не могут обеспечить надежность систем, а лишь способствуют повышению уровня надежности за счет обнаружения ошибок с помощью тестов различных категорий.

На этапе испытаний проводится системное тестирование для соответствия внешних спецификаций функций целям проекта. Испытание проводится или в реальной среде функционирования, или на испытательном стенде, который имитирует работу аппаратуры. При подготовке к испытаниям изучается "история" тестирования на процесса ЖЦ в целях использования ранее разработанных тестов, а также составления специальных тестов испытаний. На этом этапе осуществляется:

- управление ростом надежности путем неоднократного исправления и регрессионного тестирования ПО;
- принятие решения о степени готовности ПО и возможности его передачи в эксплуатацию;
- оценка надежности по результатам системного тестирования и испытаний по соответствующим *моделям надежности*, подходящих для заданных целей.

На этапе сопровождения оценка надежности ПО проводится:

- протоколирование отказов в ходе работы системы, измерения надежности функционирования и использования результатов измерений для определения потерь надежности в период времени эксплуатации;
- анализ частоты и серьезности отказов для определения порядка их устранения;

- оценка влияния функционирования системы на надежность в условиях усовершенствования технологии и применения новых инструментов разработки.

3.3. Применение моделей для оценки надежности ПО

Практика применения моделей показывает, что среди названных моделей наиболее перспективными являются модели оценочного типа, которые базируются на пуассоновских процессах (модели Мусы, Гоэла–Окомото, S-образные и др.). По этим моделям надежность стремиться к 1. Одним из недостатков является форма кривой интенсивности выявленных отказов (экспоненциальная), которая строго спускается при t>0 вблизи t=0. Это говорит о том, что при тестировании проведено недостаточно экспериментов или мало найдено ошибок, когда интенсивность отказов была близка 0. В системе остаются ошибки и их поиск требует больше времени.

В таблице 1 представлены практические значения функций интенсивности отказов $\lambda(t)$ и количество отказов $\mu(t)$ для базовых и общих моделей. В них значения a и β находятся в следующих соотношениях:

$N= a,\ \beta = a,\ b = \beta,\ \beta^1 = \beta,\ a_0 = a\beta.$

Табл. 1. Характеристика моделей надежности Пуассоновского типа
Table 1. Characteristics of Poisson type reliability models

Название модели	Функции интенсивности отказов $\lambda(t)$	Функции кумулятивного количества отказов $\mu(t)$
Модель Гоэла-Окумото	$\lambda(t) = Nb \exp(-bt)$	$\mu(t)= N(1- \exp(-bt))$
Модель Мусы	$\lambda(t) = \beta_0 \beta_1 \exp(-\beta_1 t)$	$\mu(t)= \beta_0 (1- \exp(-\beta_1 t))$
S –подобная модель	$\lambda(t) = a\beta^2 t \exp(-\beta t)$	$\lambda(t) = a\{1-(1+\beta t) \exp(-\beta t)$
Модель Шнайдевинда	$\lambda(t) = a_0 \exp(-\beta t)$	$\mu(t)= a_0/\beta (1- \exp(-\beta t))$
Общая модель пуассоновского процесса	$\lambda(t) = a\beta^{n+1} t^n \exp(-\beta t)$	$\mu(t)= a (n^1 - \sum n\beta^{n-1}/(n-1)^1 t^n \exp(-\beta t)))$

Для метода максимального правдоподобия задаются данные a, β, n, решим систему уравнений:

$$\begin{cases} \alpha = \dfrac{m}{1-\sum_{i=0}^{n} \dfrac{n!\beta^{n-i} t_m^{n-i}}{(n-i)!} \exp(-\beta t_m)} \\ \dfrac{n+1}{\beta} m = \sum_{k=1}^{m} t_k + \dfrac{m\beta^n t_m^{n+1} \exp(-\beta t_m)}{1-\sum_{i=0}^{n} \dfrac{n!\beta^{n-i} t_m^{n-i}}{(n-i)!} \exp(-\beta t_m)} \end{cases}$$

где параметр n зависит от процесса тестирования и его рекомендуемых значений:

n=0 – для небольшого проекта, в котором разработчик является также тестером (модель Мусы, Гоэло-Окомото и др.);

n=1 – для среднего проекта, в котором тестирование и проектирование ПО исполняются несколькими разработчиками из одной рабочей группы (S-образная модель);

n=2 – для большого проекта, в котором группы тестирования и проектирования работают параллельно;

n=3 – для очень большого проекта, в котором группы тестирования и разработки работают независимо друг от друга.

На основе экспериментальных данных получены функции о количестве отказов $\mu(t)$ и интенсивности отказов $\lambda(t)$ на выходных данных и значениях параметра n (рис. 4). Этот рисунок показывает вид функций $\mu(t)$ при разных значений n=0,1, 2, 3.

Наибольшее приближение достигается при n=3, а наименьшее при n=0 (модель Мусы, Гоэло-Окомото и др.). Это подтверждается соответствующими статистическими данными (табл.2), которые задают разницу между выходными данными (t_2) и соответствующими значениями функции $\mu(t)$ при значениях n = 0, 1, 2, 3.

Табл. 2. Статистические данные функции μ(t) при n=3, 2, 1 и данных t_2
Table 2. Statistical data of the function μ(t) for n = 3, 2, 1 and data t_2

Статистические показатели	**Разница функций *t_2 - μ_3***	**Разница функций *t_2 - μ_2***	**Разница функций *t_2 - μ_1***	**Разница функций *t_2 - μ***
Среднее отклонение	16.13522	16.22889	19.88387	58.93807
Медианное отклонение	15.27700	14.11600	16.0000	60.89700
Максимум отклонение	33.58100	54.23600	49.10800	88.80200
Минимум отклонение	4. 848000	-1.280000	4.175000	15.96200
Среднеквадратическое отклонение	8.374089	17.37143	14.07056	23.63765

На основе экспериментальных данных a, β, n, (табл. 2) приведены значения функций $\mu(t)$ и $\lambda(t)$ при n = 3, 2, 1, полученные при использовании методов оценки надежности Мусы, Мусы-Окомото и Шнайдевинда. Функции $\mu(t)$ для этих методов приведены на графике (рис. 4). Им соответствуют кривые экспоненциального типа. Графики этих функций близки друг другу из-за близких значений, полученных по заданным моделям.

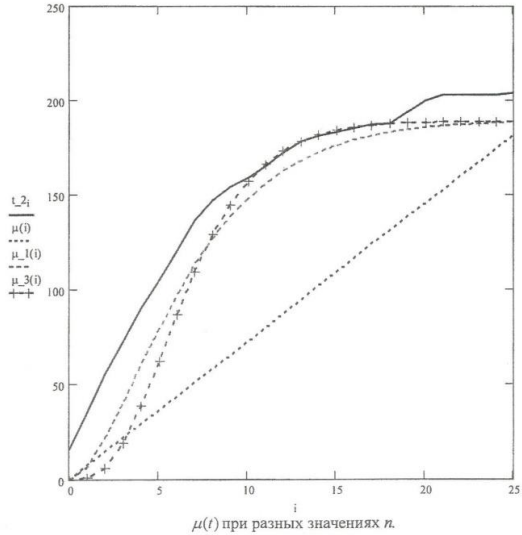


Рис. 4. Графики функций μ(t) по моделям Мусы, Окомото и Шнайдевинда
Fig. 4. The graphs of function μ(t) on models of Musa, Okumoto and Shneidewind

Для более эффективного применения приведеннях моделей надежности требуется значительное количество статистических данных о количестве и распределении отказов. А это требует увеличения количества экспериментов на процесах тестирования, системного тестирования для покрытия тестами всех компонентов и маршрутов прохождения путей в ПО.

3.4. Технологический модуль (ТМ) оценки надежности систем

ТМ разработан в рамках работ по проекту ПРОТВА ВПК (1986-1989). В состав ТМ надежности входит четыре программных модуля (ПТМ) [15, 16, 22 с.283-296]: распределение надежности, прогнозирование плотности дефектов, прогнозирование надежности и оценки надежности.

I. ПТМ «Распределения надежности» реализует метод распределения надежности по компонентам системы путем парного их сравнения и построения квадратной матрицы *A* размером $n \times n$ из элементов вида:

$$a_{11} = a_{22} = ... = a_{nn} = 1,\ a_{ij} = \frac{1}{a_{ji}}\ ,\ u, j = 1,...,n;\ i \neq j;\ n=k, l, m,$$

где n – количество сравниваемых компонентов, *k, l, m* – количество функций и модулей соответственно. Матрица включает относительный вес i-го компонента и вычисляется по формулам:

$$w_i = \frac{\sum_{j=1}^{n} a_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}}, \sum_{i=1}^{n} w_i = 1$$

В случае больших размеров матрицы в целях получения более точных относительных оценок компонентов иерархии, вычисляются, так называемые, собственный вектор и собственные значения матрицы согласно известным уравнениям [24]. В них используются следующие данные: λ_{max} – максимальное собственное значение матрицы A n–порядка, w_i – коэффициент относительного веса элементов матрицы A, W = (w_1, w_2, ... w_n) – собственный вектор, которому соответствует λ_{max}. Общность решения задачи сравнения устанавливается соотношением $\alpha = \sum_{i=1}^{n} w_i$ и значением $\sum_{i=1}^{n} w_i = 1$. Если матрица A имеет n-1 собственных значений λ, равных нулю и $\lambda_{max} = n$, то она является согласованной.

Определение индекса согласованности CI и коэффициента согласованности CR проводится по формулам:

$$CI = \frac{\lambda_{\max} - n}{n-1}, \; CR = \frac{CI}{E(CI)}, \text{ где}$$

$E(CI)$ – математическое ожидание для матрицы парных сравнений A $(n \times n)$.

Критерий приемлемости парного сравнения элементов в матрицах размером $n \geq 3$ получен такой: $CR \leq 0.05$ и $CR < 0.1$ для $n>5$. По результатам сравнения формируется квадратная матрица $F(k \times k)$.

Аналогично проводится сравнение приложений ПС. В результате сравнения получают k матриц. Возможный порядок каждой матрицы – l, а максимальный порядок каждой из них – m.

Инструмент для поддержки метода сравнения – ExpertChoice для входной матрицы A автоматически получает собственный вектор W, собственное значение λ_{max} и коэффициент согласованности CR. Для вычисления λ_{max} и W используются соответствующие функции пакета Matlab [15].

Результаты сравнений заносятся в форму, содержащую перечень весовых коэффициентов программ, критерии, индексы и коэффициенты согласованности. Они предоставляются в виде готовых результатов обработки матриц. Полученные весовые коэффициенты синтезируются с помощью пакета MATLAB 6.5. Результаты отображаются в виде отчета о распределении надежности по объектам системы.

II. ПТМ «Прогнозирование плотности дефектов» реализует набор моделей надежности для заданного класса программ системы обработки данных [15].

Прогнозирование плотности дефектов проводится по модели RLM (Rome Laboratory Model) и состоит в оценке влияния на плотность дефектов согласно следующих действий:

1) анализ значений параметров модели прогнозирования надежности, включая остаток дефектов от предыдущего этапа работ с ПО, используется для целевого распределенного значения надежности ПО;
2) сравнение прогнозируемого значения надежности с целевым распределенным значением;
3) корректировки переменных параметров для учета текущего состояния проекта ПО;
4) оценка параметров модели прогнозирования надежности;
5) прогнозирование плотности дефектов;
6) определение пороговых значений (допусков) для оценок результатов прогнозирования и анализа альтернатив;
7) расчет прогнозного значения надежности для ПО.

Полученная оценка является модификатором базового значения плотности дефектов для определенного класса ПО.

Расчет плотности дефектов делается по модели RLM (Rome Laboratory Model). Сначала выполняется однократное прогнозирование плотности дефектов по формуле:

$$D_g = \prod_{i=1}^{9} K_i,$$

где K_i – модификаторы плотности дефектов D_0, с учетом пороговых значений данных о плотности дефектов.

Затем для каждого ПО результаты сравниваются с полученными по модели RLM. Проверка показала, что для ПО объемом 10 - 25 KSLOC погрешность прогнозировании плотности дефектов – примерно составляет 30-35%. Это объясняется некоторыми ограничениями системы Hugin Lite 6.5. Полученные результаты по определению плотности дефектов используются при прогнозировании надежности ПО.

III. ПТМ «Прогнозирование надежности» реализует метод прогнозирования значения надежности по каждому модулю системы по следующей модели надежности [15, 23]:

$$R_i = \exp[-D_i\, I_i \cdot (1-\exp(\frac{\rho_i \cdot K}{I_i \cdot \varphi_i} \cdot t))],$$

где ρ_i – параметр среды эксплуатации i-го модуля, φ_i – характеристика среды ее разработки, I_i – оцененный размер начального кода, а D_i – прогнозируемая плотность дефектов в системе. Коэффициент дефектов K – константа, предвиденная для всех объектов ПС, а значения ρ_i и φ_i – известны на момент

первоначального прогнозирования надежности, они не изменяются во время разработки компонентов системы.

IV. ПТМ «Оценка надежности системы» согласно классификации дефектов (Orthogonal Defect Classification), в соответствии с которой для каждого выявленного дефекта определяются параметры: тип дефекта, триггер дефекта, влияние дефекта. Эти параметры используется одной или двумя подходящими моделями надежности из выше приведенного в целях проведения оценки прогнозного значения надежности отдельных модулей и системы в целом. Результаты оценки сравниваются, и выбирается из них наиболее правдоподобная модель.

По стандарту ISO/IEC 9126 (1-4) определяются показатели качества (табл. 3).

Табл. 3. Характеристики качества в стандарте ISO/IEC 9126
Table 3. Characteristics of quality in ISO/IEC 9126 standard

№	Наименование характеристики	Определение характеризуемых свойств ПС
q1	Функциональность (functionality)	Свойства ПП, обусловливающие способность выполнения функций в соответствии требованиям в процессе тестирования и испытания системы в заданной среде
q2	Надежность (reliability)	Свойства ПП, обусловливающие ее способность сохранять уровень функционирования и низкую вероятность отказов в процессе выполнения
q3	Применимость (usability)	Свойства ПП, обусловливающие ее способность быть понимаемой и удобной для использования в указанных условиях
q4	Эффективность (efficiency)	Свойства ПП для рационального использования выделенных ресурсов при работе системы в установленных условиях
q5	Сопровождаемость (maintainability)	Свойства ПП, которые обеспечивают модификацию, усовершенствование или адаптацию системы к изменениям среды, требований и функциональности.
q6	Переносимость (portability)	Свойства ПП, обусловливающие ее способность быть перенесенным из одной среды в другую.

На основе полученных данных о надежности и других показателях качества (функциональность, эффективность и др.) рассчитывается целевое значение завершенности и полезности системы (ПС), адекватных потребностям заказчика. При этом мера эксплуатационного качества системы определяется функцией полезности вида:

$$Q_{nc} = \sum_{i=1}^{k} a_i \cdot R_i$$

где a_i – мера важности *i*-й функции системы для процесса, R_i – надежность выполнения функций в заданном периоде *t* эксплуатации системы.

Данные по всем показателям качества (q-quality) q_1-q_6 в табл. 3 оцениваются по формуле:

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

где a_i - атрибуты каждого показателя качества (*i*=1-6); m_{ij} – метрики q_i показателя с *j*-атрибутами качества; w_{ij} - вес *i* -показателя качества системы с *j*-атрибутами. Полученные данные по показателям (характеристикам) модели качества и R_i – надежность выполнения функций входят в сертификат качества [15].

4. Заключение

В работе рассмотрены подходы к оценке надежности технических и программных систем с применением моделей надежности из множества существующих моделей разных видов и типов. Определены основные базовые понятия надежности, обеспечивающие оценку надежности по соответствующим моделям надежности ПС, основанным на времени функционирования и/или количестве отказов (ошибок), получаемых в компонентах в процессах ЖЦ тестирования, системного тестирования и эксплуатации системы. Согласно приведенной классификации моделей надежности процессы обнаружения ошибок в программах носят случайный Марковский и пуассоновский характер и обеспечивают поиск ошибок, дефектов и отказов.

Некоторые модели надежности позволяют прогнозировать число ошибок в процессе тестирования, другие оценивать надежность с помощью функций надежности по данным, собранным на этапах ЖЦ разработки системы и испытания. Для примера приведены экспериментальные данные для оценки интенсивности отказов λ *(t)* и количества отказов μ*(t)* с помощью базовых (Мусы, Гоэла-Окомото и др.) и общей модели надежности, собранных данных на этапах ЖЦ и приведены сравнительные оценки результатов оценки.

Дано описание инструментального комплекса модулей ПТМ, обеспечивающих распределение надежности, прогнозирование плотности дефектов и оценки надежности. Приведены показатели качества в стандартной модели ISO 9126 (1-4) и оценки качества, включая измерение показателя надежности, а также других показателей качества, которые входят в сертификат готового продукта.

Список литературы

[1]. Липаев В.В. Надежность программного обеспечения. М.: СИНТЕГ, 1998, 231 стр.

[2]. Липаев В.В. Методы обеспечения качества крупномасштабных программных систем. М.: СИНТЕГ, 2003, 510 стр.
[3]. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980, 360 стр.
[4]. Мороз Г.Б., Лаврищева Е.М. Модели роста надежности программного обеспечения. Киев: Институт кибернетики АНУ, препринт 92–38, 1992, 23 стр.
[5]. Липаев В.В.. Надежность и функциональная безопасность комплексов программ реального времени. Москва, ЗАО «Светлица», 2013, 193 стр.
[6]. Shick G.J., Wolverton R.W. An analysis of computing software reliability models. IEEE Transactions on Software Engineering, vol. SE–4, № 2, 1978, pp. 104–120.
[7]. Shanthikumar J.G. Software reliability models: A Review. Microelectronics Reliability, vol. 23, № 5, 1983, pp. 903–943.
[8]. Goel Amrit L. Software reliability models: Assumptions, limitations, and applicability. IEEE Transactions on Software Engineering, vol. SE–11, № 12, 1985, pp. 1411–1423.
[9]. Musa J.D. Okumoto K. A. Logarithmic Poisson Time Model for Software Reliability Measurement. In Proc. of the 7th International Conference on Software Engineering, 1984, pp. 230–238.
[10]. Yamada S., Ohba M., Osaki S. S–shaped software reliability grows modeling for software error detection. IEEE Transactions on Reliability, vol. R–32, № 5, pp. 475–478.
[11]. Chulani S. Constructive quality modeling for defect density prediction: COQUALMO. In Proc. of the International Symposium on Software Reliability Engineering (ISSRE'99), 1999.
[12]. Гнеденко Б.В., Коваленко И.Н. Введение в теорию массового обслуживания. М., Наука, 1966, 432 стр.
[13]. Коваленко И.Н., Шпак В.Д. Вероятностные характеристики сложных систем с иерархическим управлением. Известия АН СССР. Техническая кибернетика, no. 6, 1972, стр. 30-34.
[14]. Duval P., Matyas R., Grover A. Continuous integration improving Software quality and reducing risk. Addison Wesley, 2009, 691 p.
[15]. Коваль Г.И. Модели и методы инженерии качества систем на ранних этапах ЖЦ. Реф. дис. ИК НАНУ, 2005, 20 стр.
[16]. Андон Ф.И., Коваль Г.И. и др. Основы инженерии качества программных систем. К.: Наукова думка, 2007, 670 стр.
[17]. Горбенко А.В., Засуха С.А., Рубан В.И., Тарасюк О.М., Харченко В.С. Безопасность ракетно-космической техники и надежность компьютерных систем: 2000-е годы. Авиационно-космическая техника и технология, №1(78), 2011, стр. 9-20.
[18]. A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11-33.
[19]. IEC 62628. Guidance on software aspects of dependability. Geneva: IEC, 2011, 63 p.
[20]. ISO 15288:2002. Systems Engineering. Cycle Life Processes of Systems.
[21]. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика. К.: Наукова думка, 2006, 452 стр.
[22]. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. К.: Наукова думка, 2009, 372 стр.
[23]. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, технологии, CASE-средства. К.: Наукова думка, 2014, 284 стр.

[24]. Саати Т. Принятие решений. Метод анализа иерархий. М.: Радио и связь, 1993, 315 стр.

Analysis of methods for assessing the reliability of equipment and systems. Practice of methods

[1,2]*E.M. Lavrischeva <lavr@ispras.ru>*
[1,2,3]*N.V. Pakulin<npak@ispras.ru>*
[1]*A.G. Ryzhov <ryzhov@ispras.ru>*
[1,3]*S.V. Zelenov <zelenov@ispras.ru>*
[1] *Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia,
[2]*Moscow Institute of physics and technology (MIPT)*
141700, Russia, Moscow region, Dolgoprudny, Campus per., 9.
[3] *National Research University Higher School of Economics (HSE)*
20 Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. The analysis of models and methods of reliability evaluation of hardware and software is carried out. The basic concepts of reliability and safety methods of such systems and situations leading to errors, defects and failures are defined. The definition of reliability and safety of technical systems and software systems is given. The classification of reliability models: predictive, measuring and evaluation types. Evaluation models that are used more in practice are described. The standard of Software life cycle (ISO 15288:2002) is defined, focused on the development and control of system components for errors, starting with the system requirements. The results of application of reliability models (Moussa, Goel-Okomoto, etc.) to small, medium and large projects are presented and their comparative assessment is given. The technological module (TM) of reliability evaluation of complex software systems VPK (1989) is described. The quality model of the standard ISO 9126 (1-4): 2002-2004 with indicators of functionality, reliability, efficiency, etc., which are used in determining the maturity and certificate of the product is shown.

Keywords: reliability, model, method, error, defect, failure, random process, safety, dependability, recoverability, fault tolerance, completeness, reliability assessment, quality certificate.

DOI: 10.15514/ISPRAS-2018-30(3)-8

For citation: Lavrischeva E.M., Pakulin N.V., Ryzhov A.G., Zelenov S.V. Analysis of methods for assessing the reliability of equipment and systems. Practice of methods. *Trudy ISP RAN/Proc. ISP RAS*, том 30, вып. 3, 2018 г., стр. 99-120. DOI: 10.15514/ISPRAS-2018-30(3)-8

References

[1] Lipaev V. V. Software Reliability. M.: SINTEG, 1998, 231 p. (in Russian)
[2] Lipaev V. V. Methods of quality assurance of large-scale software systems. M.: SINTEG, 2003, 510 p. (in Russian).
[3] Myers G. Software Reliability, M.: Mir, 1980, 360 p. (in Russian).

[4] Moroz G. B., lavrisheva E. M. Models of software reliability growth. K.: V.M. Glushkov Institute of Cybernetics of NAS of Ukraine, preprint 92-38, 1992, 23p. (in Russian).
[5] Lipaev V. V. Reliability and functional safety of software systems real time. Moscow, Svetlitsa, 2013, 193 p.
[6] Shick G.J., Wolverton R.W. An analysis of computing software reliability models. IEEE Transactions on Software Engineering, vol. SE–4, № 2, 1978, pp. 104–120.
[7] Shanthikumar J.G. Software reliability models: A Review. Microelectronics Reliability, vol. 23, № 5, 1983, pp. 903–943.
[8] Goel Amrit L. Software reliability models: Assumptions, limitations, and applicability. IEEE Transactions on Software Engineering, vol. SE–11, № 12, 1985, pp. 1411–1423.
[9] Musa J.D. Okumoto K. A. Logarithmic Poisson Time Model for Software Reliability Measurement. In Proc. of the 7th International Conference on Software Engineering, 1984, pp. 230–238.
[10] Yamada S., Ohba M., Osaki S. S–shaped software reliability grows modeling for software error detection. IEEE Transactions on Reliability, vol. R–32, № 5, pp. 475–478.
[11] Chulani S. Constructive quality modeling for defect density prediction: COQUALMO. In Proc. of the International Symposium on Software Reliability Engineering (ISSRE'99), 1999.
[12] Gnedenko B.V., Kovalenko I.N. Introduction to the queueing theory. M.: Science, 1966, 432 p. (in Russian).
[13] I. N. Kovalenko and V. D. Shpak. Probabilistic characteristics of complex systems with hierarchical control. Izv. Akad. Nauk SSSR, Tekhn. Kibern., no. 6, 1972, pp. 30–34 (in Russian).
[14] Duval P., Matyas R., Grover A. Continuous integration improving Software quality and reducing risk. Addison Wesley, 2009, 691 p.
[15] Koval G.I., Models and methods for engineering quality systems at early stages of life cycle. IK NANU, Extended abstract of PhD Thesis, 2005, 20 p. (in Russian)
[16] Andon F. I. et al. Foundation of quality engineering software system. K.: Naukova Dumka, 2007, 670 p. (in Russian)
[17] Gorbenko A.V.et al. the Safety of rocket-space engineering and reliability of computer systems. Aerospace technics and technology, №1 (78), 2011, pp. 9-20, 2011.
[18] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11-33.
[19] IEC 62628. Guidance on software aspects of dependability. Geneva: IEC, 2011, 63 p.
[20] ISO 15288:2002. Systems Engineering. Cycle Life Processes of Systems.
[21] Lavrischeva E. M. Programming Methods. Theory, engineering, practice. K.: Naukova Dumka, 2006, 452 p. (in Russian).
[22] Lavrisheva E. M., Grishchenko V. N. Assembly programming. Foundation of software industries. K.: K.: Naukova Dumka, 2009, 372 p. (in Russian).
[23] Lavrisheva E. M. Software Engineering of computer systems. Paradigms, technologies, CASE-means. K.: K.: Naukova Dumka, 2014, 284 p. (in Russian).
[24] Saati T. Decision-Making. Method of hierarchy analysis. M.: Radio and communications, 1993. 315p. (in Russian).

In-Kernel Memory-Mapped I/O Device Emulation

[1,4]*V.Yu. Cheptsov<cheptsov@ispras.ru>*
[1,2,3,4] *A.V. Khoroshilov<khoroshilov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of RAS*
25 Alexander Solzhenitsyn Str., Moscow, 109004, Russia
[2] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[3] *Moscow Institute of Physics and Technology,*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4] *Higher School of Economics.*
20, Myasnitskaya Ulitsa, Moscow 101000, Russia

Abstract. Device emulation is a common necessity that arises at various steps of the development cycle, hardware migration, or reverse-engineering. While implementing the algorithms behind the device may be a nontrivial task by itself, connecting the emulator to an existing environment, such as drivers intended to work with the actual hardware, may be no less complex. Devices relying on memory-mapped input/output are of a particular interest, because unlike port-mapped input/output there is much less of a chance that the target platform provides a direct interface to intercept the transmissions. A well-known approach used in various virtual machine software is to put the entire operating system under a hypervisor and build the emulator externally. This may not be desirable for reasons like hypervisor complexity, performance loss, and additional requirements for the host hardware. In this paper we extend this approach to the kernel and explain how it may be possible to build the emulator by relying on the existing interfaces provided by an operating system. Given the common availability of an MMU unit as well as memory protection mechanisms, allowing the handling of page or segment traps at read or write access, we presume that a suggested technique of intercepting memory-mapped input/output could be implemented in a broad number of target platforms. To illustrate the specifics and show potential issues we provide the ways to simplify the implementation and optimize it in speed depending on the target capabilities, the protocol emulated, and the project requirements. As a working proof we created a SMC emulator for an x86 target, which makes use of this approach.

Keywords: device emulation; memory-mapped i/o; kernel modules

DOI: 10.15514/ISPRAS-2018-30(3)-9

For citation: Cheptsov V.Yu., Khoroshilov A.V. In-Kernel Memory-Mapped I/O Device Emulation. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 121-134. DOI: 10.15514/ISPRAS-2018-30(3)-9

1. Introduction

One of the common engineering demands is device emulation. It may arise during the software development cycle, for example, in testing or driver verification, at hardware migration, when there is no easy way to rewrite the existing software. Other than that, in the world of proprietary hardware and software it is not rare that the only way to understand and document the device abilities is to reverse-engineer it, and the ability to dynamically debug or reverse-engineer the code could be the key in security analysis or adding the device support to a virtual machine.

Speaking of virtual machines, or rather hypervisors, building the entire virtual stack for a single device one needs to emulate is often an overkill due to performance reasons, although it could be partially mitigated by hardware-assisted virtualization and software compatibility. The latter may involve working on completely unrelated parts of the driver stack and result in unnecessary costs for continuous support.

However, while the development of full platform emulators is a considerably common topic with abundance of existing papers and products like qemu, bochs, iOS simulator, etc., peripheral emulation is much less widespread. In some cases, virtual machine guest tools do try to mimic certain hardware, but even that is usually implemented as a part of a full scale platform emulation. The problem with the peripherals is not just in implementing the algorithms behind the device, which may be a nontrivial task by itself, but also connecting the emulator to an existing environment, such as other drivers above in the stack intended to work with the real hardware.

Since one of the important aspects of using any peripherals is the ability for the CPU to communicate to them, the common demand for a device emulator is to provide a way to do it. Presently there are two common low-level approaches to perform input and output operations: port-mapped I/O (PMIO) and memory-mapped I/O (MMIO). While there are other ways such as involving some dedicated hardware, they are relatively less widespread. High-level communications operating on a packet basis (like USB bus) usually go through the dedicated abstraction layer, and thus may be implemented with the standard APIs offered by the operating system without any special effort.

It is fairly easy to implement communication protocols with a hypervisor, the standard approach is to ensure that accessing certain memory exits the virtual machine context (vmexit), which is later handled by the implementation. However, as we mentioned previously, the use of a hypervisor may be impractical, and we have to look for other means of intercepting memory access. Since direct memory access is very common, yet quite problematic to intercept, in this paper we explain how one could implement a considerably portable MMIO emulator in the kernel and cover the details of emulating device communication protocols on common platforms.

2. State of the art

We admit to not being the first to experiment with peripheral device emulation. Every single year several published papers in the field of hardware virtualisation cover this topic to a certain level. Articles published by VMware Inc. researchers [1] [2] provide an in-depth coverage of x86- compatible hardware emulation. They explain the existing obstacles and necessary actions to be taken to implement a complete virtual stack from the CPU to network adapters. In their works they pay a lot of attention to performance optimization, hardware-assisted virtualization and show a visible performance penalty reduction over the new CPU generations in Virtualization nanobenchmarks section of the first referenced paper.

As a result of continuous contribution from different parties and competitive product development, the general hypervisor performance has dramatically improved. While GPU emulation is out of the scope of this paper, it should be admitted that there are several works which do manage to provide a complete GPU emulation at a reasonable performance [3] [4]. These works feature an open GPUvm platform in the Xen hypervisor.

Another related direction involves security analysis or reverse-engineering. While less frequently found in academic writing, there are several products, tools, and patches for Linux intended to log execution details from the Linux kernel for later analysis. One of the most well-known toolsets is Linux Trace Toolkit, and one of the most prominent cases of applying the approach in practice is for Nouveau driver development for NVIDIA GPUs. Enabling OS Research by Inferring Interactions in the Black-Box GPU Stack by Konstantinos Menychtas, Kai Shen, and Michael L. Scott [5] provides a good coverage in detail.

3. Basic I/O Introduction

Port-mapped I/O is usually more demanding to the CPU instruction architecture and requires a number of so-called ports the devices will be mapped to, and perhaps a dedicated instruction set to access these ports as well. Because the device memory is accessed indirectly, another name for PMIO is detached I/O.

As an example, one of the most popular architectures to implement PMIO is x86. It can be utilized by means of two dedicated instructions: in and out, which enable one to receive and send 8, 16, or 32 bits of data to a port from 0 to 65535. Since there are faster ways to perform I/O on x86 and PMIO is not recommended for use nowadays, in some literature it may be referred to as legacy I/O. This may not be the case for other architectures found in micro-controllers, but in general MMIO support is increasing.

Memory-mapped I/O involves direct mapping of the device memory to the host memory, enabling the software to access the device just like a normal chunk of noncacheable RAM with the use of the native instruction set. Since MMIO implementation is often faster than PMIO and sometimes simpler to use, it will be the one to opt for when implementing a device communication protocol. For example, on

x86 various devices installed as PCI extension cards or system management controllers make a use of it.

Virtual devices are not supposed to be functionally different from real hardware. For this reason, emulators have difficulties supporting I/O communication protocols. The taken approach varies depending on the demands and available resources, but usually one of the following is used:

- Custom device development
- Driver reimplementation
- Building a hypervisor

Sadly, each of these has serious limitations, and most of them create obstacles for generic peripheral emulation, as observed in Table. 1.

Table 1. Pros and cons summary

	Device	Driver	Hypervisor
Software independency	+	-	±
Low costs	-	±	+
Legal issues	+	-	-
Infrastructure dependency	-	+	-
Forward compatibility	+	-	-
Performance	+	±	±
Other device support	+	+	-

Developing a new device by extending a microcontroller to offer a required interface or creating an entire chip mostly works for very simple devices when a single copy is going to be used for some kind of deep debugging or instrumentation. A good example could be removable BIOS chips for debugging or HDMI to VGA adapters with HDCP decoding. While this solution is very reliable for creating a test device, the results of mass-producing a customised device will likely be not worth the effort. It will be either more expensive or worse in quality. In addition, it is important to have the legal part of the question in mind and avoid patent infringement. However, this method could be most reliable when it comes to stability.

Reimplementing the driver to support another communication interface for the virtual device is very useful when working with performance-critical hardware such as GPUs. For them each extra communication layer may heavily affect the performance due to high bandwidth usage, and that is why virtualization software implements extended GPU support (like DirectX or OpenGL) in such a way. However, in our case it defeats the entire purpose of creating a virtual device. If the point is to test the driver, it will no longer stay the same. If the reason is to support a proprietary driver, one will have to reverse-engineer it and have issues every time it gets updated.

Bringing in a virtual machine with a hypervisor is a way to overdo it. While a decent virtual machine has a wide range of supported hardware, it adds a lot of downsides as well. In particular there will always be potential performance issues, even with hardware-assisted virtualization support. More than that, compatibility issues will

likely become a blocker if the rest of the environment is not generic and well-known. It is unfortunate, but even the mainstream operating systems may be unwilling to expose new interfaces for virtual machines (like most of the graphical stack on Apple macOS).

4. Intercepting the I/O

As a result I/O interception comes out as a pragmatic way to achieve the goal. Despite not being very common, software and hardware have enough capabilities to intercept raw device communication without touching the higher-level drivers themselves.

For example, for the past 8 years the recent x86 firmwares contain a dedicated UEFI System Management Mode [6] protocol to intercept PMIO. This protocol originally existed as a EFI_SMM_IO_TRAP_DISPATCH_PROTOCOL protocol[1], but later on was extended with an additional IO_TRAP_EX_DISPATCH_PROTOCOL protocol[2]. Both protocols allow you to create direct handlers to intercept the portmapped access. By design, the management mode affects the operating system code as well, so it works throughout the boot process and is fully transparent to the higher level software implementations like OS kernel or drivers. However, aside from not being very well documented, third-party code execution in the System Management Mode is generally prohibited. So even if one is to reimplement the SMI handler similar to what Intel offers with the open source platform code, it will be of no use for anyone but UEFI firmware developers.

Fortunately, most of PMIO interface code is usually well abstracted in the kernel, and when it comes to intercepting you could just replace the underlying low level function implementation within the emulator context. However, devices relying on MMIO are of a particular interest, because unlike PMIO there is a much less chance that the target platform provides a direct interface to intercept the transmissions.

For embedded devices it may well be sufficient to statically analyze the firmware, find the instructions responsible for I/O, and either dynamically or statically overwrite them to jump to prepared thunks that will handle them accordingly. This approach is common for security analysis especially when very little is known not only about the explored peripherals but the whole controller. However, since the firmware or the driver may receive updates in the future, this approach is not very effective outside of security or code coverage analysis, and the like.

One of the first ideas that comes to mind due to the nature of MMIO writes is relying on CPU debug registers. These registers (e.g. DR on Intel or BP_CTRL/BP_COM on ARM Cortex) allow you to implement hardware breakpoints or rather watchpoints, which may trap read and write access. However, these registers are very few, and their scope area is small (i.e. a 32-bit or 64-bit word). Other than that, the kernel,

[1] GUID: 58DC368D-7BFA-4E77-ABBC-0E29418DF930
[2] GUID: 5B48E913-707B-4F9D-AF2E-EE035BCE395D

debuggers, or other software may use these registers for their own needs, which leads to them being simply impractical for this kind of work.

In general-purpose operating systems with defined kernel APIs there are much better ways, such as a page protection mechanism, which is used to implement watchpoints in software. While this is suitable for doing MMIO emulation, most of the known works relying on this technique either use it for tracing or just for debugging backends. The notable example is MMIO trace in Linux, which was originally developed to reverse-engineer proprietary NVIDIA drivers by tracing the register access [7]. Other than that, there are very few examples of how it can be utilized for device I/O emulation.

5. Proposed approach

The idea of general purpose I/O interception is very simple: catch reads and writes, make sure that the values read are correct, and the values written are accounted for. To apply it to MMIO we could limit page protection of the target area, and trap the faults as they happen. Due to bandwidth limitations and architecture simplicity the I/O sequences are generally serialized, even if they happen from different threads. It may not be the case for GPUs, yet GPUs likely will not need this kind of emulation due to performance reasons. Still, in general if serialized I/O is not guaranteed even within a single memory page (which is rare) one could always implement it manually by utilizing the synchronization primitives.

Therefore, the most obvious approach will be:

1. mark the relevant page as neither writable nor readable (not present in x86 terms);
2. catch a fault and decode the fault address and the direction (in or out);
3. disassemble the instruction that caused the fault and obtain its operands from the frame;
4. handle the operands for the emulation;
5. update the destination registers or memory for the reads as necessary;
6. return to the location after the instruction, which caused the fault.

While it indeed solves the problem and looks very straightforward, the implementation itself could be very convoluted. While the saved context is likely to contain the fault and return addresses, bringing a full-scale disassembling framework to the kernel is inflexible due to extra architecture dependencies and considerable amounts of code required for instruction emulation. Even more, it may impose additional performance penalties, which are already tough enough.

For these reasons we tried to alter the algorithm in a way that would be simpler, less platform-dependent, and similarly performant. After examining several real-world examples, we consider the following model of a MMIO-based I/O protocol, which could be applied to quite a number of devices:

1. host ensures that the target is ready for an I/O operation;

2. host performs the I/O operation (by reading or writing at a defined address space);
3. host ensures that the operation is complete and repeats the process.

The 1st and 3rd steps are usually implemented as a write-and-poll, a write-and-interrupt or just as a poll. Another advantage comes out from common differences in frequencies between the host and the peripheral. Since communications usually happen between the devices with different clock bases, most of the protocols are synchronous, and the host generally does not overwrite the areas it has just written to without making a read to confirm it was successful. Even more, most of the protocols are stateful, and it is uncommon to see subsequent reads from the same place expecting the value to change more than once. A write operation will most likely appear in-between.

Under these assumptions we use a simple satisfactory transaction model as an example:

1. write operation type (read or write);
2. read acknowledge status until status ready;
3. handle the values:
   3.1. read the value for read operations;
   3.2. write the value for write operations and read acknowledge status until status ready.

5.1 With write-only page support

If write-only pages are supported, in a number of cases one may implement a flip-flop approach that will switch page protection from read-only to write-only and backwards as the process goes.

To emulate the proposed transaction we could start the communication process with the page marked as read-only, which will then trap on operation type. Here we will initiate the transaction and switch the protection to write-only. After the operation is written the trap on the status read will trigger, where we will read the written operation type, update the value for read operations and set its status. Afterwards the page protection is returned to read-only and the control is transferred back to the driver. For read operations that is all of it, for write operations the driver will read the status and attempt to perform the actual write, which should trigger the trap again. From there on one could repeat the process as described for the operation type. In the end for both reads and writes page protection returns back to read-only, eliminating any platform-specific disassembling and relying on generic approach.

Expectedly one does not have any easy access to write-only pages on popular architectures such as ARM [8] or x86. Perhaps, if these architectures were originally designed at present, when the demand for better memory protection management is much higher and when features like W^X memory and execute-only memory have already become commonplace, we would have had finer memory management that would support write-only pages. However, nowadays write-only pages are not very

common in both hardware and software implementations. Certain PowerPC implementations [9] or processor extensions may provide access to them, so it remains a good idea to check CPU manuals before abandoning the try. For example, Intel x86 processors starting from Nehalem technically support write-only memory via EPT (Extended Page Tables [10]), yet it can hardly be used for anything but virtualization.

5.2 Without write-only page support

When write-only pages are not available, we may still be able to work out a simpler approach, and this is where memory patching comes in hand. The idea is to let the original instruction perform the I/O just as normal, but to encode a jump-back instruction right afterwards to ensure that page protection is limited again to trap the next I/O operation. Initially this approach may appear to have too many issues to be considered in practice, however, they could all be solved with enough effort, and some of them could even be turned into benefits.

The first issue to solve is the length of the faulted instruction. A number of architectures provide fixed-length instruction sets, so the next instruction address to encode our jumpback instruction could be calculated even without knowing anything about the current instruction. For others one could write or find simple instruction fetchers, to only decode the length without operand or operation details. Such software may also go under the name of length disassemblers, and various implementations exist for popular platforms [11]. It may become a little more involved when the I/O instruction results in non-linear control flow, but in general I/O and branching instructions belong to separate classes and are not mixed together.

The second issue occurs when the device memory is mapped to userspace and the communication happens in userspace as well. In this case a direct jump to protection restoration code is not possible, and a breakpoint or similar instruction will have to be encoded to trigger the context switch, return to the kernel and pass the control to our handler.

The third and probably the most serious issue happens when I/O operations are performed through shared code. By assuming serialized I/O we consider no cases of simultaneous code execution from the same area (unless there are multiple devices). Therefore, we could safely patch it. However, nothing prohibits the driver from utilizing generic memory primitives like memcpy or memset to bulk-write or read the dedicated area. These primitives generally have no effect on the I/O itself, and we do not need to intercept every byte they touch. To avoid the issue one could examine the stack trace and modify the instruction at the return address. Not only this does not require disassembling but also reduces the penalty from trapping extra I/O operations, so a quick stack unwinding that can often be implemented with compiler intrinsics easily pays off.

With all the pieces put together it creates a solid approach for a large chunk of I/O protocols. In addition to these general improvements platform-specific optimizations could be applied. For example, extra page protection changes may be avoided for

write operations, if the hardware may ignore interrupts caused by write protection violation (CR0 WP bit on x86). It should be noted that one is to pay extra attention to the scheduler (e.g. disable preemption) not to let it switch the task to another core, where write protection is on.

6. Evaluation

To apply the proposed solution in practice we created a software-based emulator for the 2nd generation Apple SMC in a form of a kernel extension for Apple macOS. System Management Controller (SMC) is a chip commonly found in Intel-based Apple Macintosh computers or certain Google Chromebooks. This chip is responsible for computer power management, display backlight control, HDD monitoring, thermal control, hybrid sleep and hibernation support, external device current regulation (AirPort, USB, FireWire), charging the battery, trackpad controls, screen mirroring, etc. This chip is not essential for computer functioning, and could be viewed as a convenience feature for a vendor to rely on to centralize and simplify hardware management.

There are two main generations of SMC controllers in Apple computers. The 1st generation was built on a 16-bit Renesas H8S/2117 controller and exposed port-mapped I/O interfaces to communicate with the operating system. The 2nd and subsequent generations are based on 32-bit ARMv7-A processors, and expose memory-mapped and port-mapped I/O interfaces. Both approaches are used to implement the same functionality within a single synchronous stateful protocol. Initially the communication happens via the PMIO protocol, and then a switch to MMIO protocol happens if the device supports it. The whole communication process happens within the kernel and the existing drivers for the 2nd generation hardware are closed-source. Fortunately, due to side researchers the communication protocols are mostly documented [12].

The reasons for taking this particular device into consideration was not only because it is a challenging task compared to devices with open specifications and decent documentation, but also for the importance of having better control of the hardware you use. Apple SMC has complete access to every device in the system and could monitor the bus communications. Other than that it stores temporary encryption keys for hibernation images or user action free restarts (authenticated restarts), when full disk encryption is enabled. Apple SMC drivers expose a dedicated protocol to userspace. This protocol provides a way to obtain SMC data and configure both SMC and onboard devices. Given its direct connection to the hardware, it may be possible to inflict damage on the computer by overheating or causing power surges. Moreover, previous researches discovered that it was very easy to modify SMC firmware, which is also a very serious concern [13].

The actual implementation follows the proposed approach without write-only page support with all the suggested optimizations and certain platform-specific adjustments. SMC MMIO protocol covers a 64 KB area, which we split into pages

with the dedicated handlers based on the page index. Since the access to each page is serialized, no additional I/O wrapping is necessary.

In the XNU kernel, which powers all modern Apple hardware including Macs, Intel CPU exceptions are routed through a dedicated kernel_trap function. To let the driver communicate with the emulated device we added a SMC nub via the standard I/O Kit APIs with mapped memory regions with restricted protection and extended the kernel_trap function in EXC_I386_PGFLT handling code specifically for our memory.

A simplified version of this code is shown in Listing 1. ioRegionStart and ioRegionEnd locate the emulated I/O area starting and ending addresses, appleSmcStart and appleSmcEnd point to the AppleSMC driver address range. instrSize function calculates the instruction length at the return address to later write the jump-back code via writeTrampoline function, which not only writes the trampoline code (by disabling the WP bit and interrupts) but additionally disables CPU preemption to avoid the scheduler switch.

```
auto faultAddr = state->cr2;
if (faultAddr >= ioRegionStart &&
    faultAddr < ioRegionEnd) {
  auto retAddr = state->rip;
  if (retAddr >= appleSmcStart &&
      retAddr < appleSmcEnd) {
    // Simple case (from AppleSMC)
    retAddr += instrSize(retAddr, 1);
  } else {
    // Complex case (from e.g. memcpy)
    retAddr = unwindToSMC(state->rsp);
  }

  auto faultType = FaultTypeRead;
  if (state->err & T_PF_WRITE) {
    faultType = FaultTypeWrite;
  }
  updateProtection(faultType, faultAddr);
  saveOrgCode(retAddr, TrampolineSize);
  writeTrampoline(faultType, faultAddr);
  return;
}
```

Listing 1. Sample code

To transfer the control flow to the protocol emulator updateProtection is performing the actual protection upgrade of the emulated I/O area and invokes the read access handler. It should be noted that a dedicated procedure may be needed for platforms with delayed physical mapping update. For example, with XNU it is necessary to trigger virtual memory fault twice when the page is not present. Similarly, the

protection restoration routine invoked from the trampoline preserves the registers and calls the write handler.

As a result, it was possible to emulate all the existing SMC protocols at no issue and avoid the use of the original device.

7. Conclusion

Emulating peripheral devices within the existing operating system is not a new problem. Different solutions and approaches have appeared over the years. The industrial demand for full-stack operating system virtualization brought their performance to a completely different level, and the needs for better customization resulted in operating system developers providing more flexible interfaces with the possibility to create virtual hardware out of the box. Programmable microcontrollers made the process of building a device clone with the necessary features a much simpler task to accomplish.

However, there are numerous cases, where in-kernel peripheral emulation is highly anticipated, such as driver development needs, testing and verification, hardware migration, security analysis, etc. As we stated, it is often not possible or extremely impractical to attempt to incorporate virtual machines due to development costs or performance penalties. While virtual machines succeed in emulating CPUs of the same architecture at almost the same speed with hardware assisted virtualization, the performance of other CPUs without the use of JITs, commonly used in video game console emulators but rarely found in generic virtualization software, is often much worse. And in terms of I/O emulation, which is the primary concern of this paper, the situation is no better.

Furthermore, all the solutions heavily depend on the target architecture. While it was possible to think of x86 as the main architecture for personal computers in the beginning of 2000- s, today the concept of personal computers has shifted away, and other major players, e.g. ARM, appeared on the market. With this in mind the classical approach to virtualizing the whole operating system could face severe issues in the future.

The idea of using page protection faults to handle device I/O events without a hypervisor may be known but not widespread anywhere out of I/O tracing. In this paper we described a way to implement a complete MMIO protocol emulator in the kernel with the use of a generic approach that has few dependencies on the target architecture and relies on platform features such as MMU and paging. We showed that certain target architecture capabilities and device protocol specifics may affect the implementation, and effectively allow or disallow a broad range of optimizations. We believe that a suggested device I/O protocol model is applicable to various hardware, and give examples on how to simplify and optimize its implementation. After exploring the existing hardware, we built a SMC emulator in the XNU kernel to illustrate the suggested approach.

Acknowledgements

The authors thank ISP RAS and SYRCoSE staff for review and comment, Nikita Golovliov for aid in SMC emulator development, and Marvin Häuser for reverse-engineering Apple SMC UEFI drivers.

References

[1]. Jeremy Sugerman, Ganesh Venkitachalam, Beng-Hong Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In Proceedings of the General Track: 2001 USENIX Annual Technical Conference, 2001, pp. 1-14. Available at: http://static.usenix.org/legacy/publications/library/proceedings/usenix01/sugerman/sugerman.ps, accessed 12.06.18

[2]. Keith Adams, Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. In Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, 2006, pp. 2-13. Available at: https://www.vmware. com/pdf/asplos235 adams.pdf, accessed 09.06.18

[3]. Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji. GPUvm: Why Not Virtualizing GPUs at the Hypervisor? In Proceedings of the 2014 USENIX Annual Technical Conference, 2014, pp. 109-120. Available at: https://www.usenix.org/system/files/conference/ atc14/atc14-paper-suzuki.pdf, accessed 12.06.18

[4]. Hangchen Yu, Christopher J. Rossbach. Full Virtualization for GPUs Reconsidered. In Proceedings of the Annual Workshop on Duplicating, Deconstructing, and Debunking, 2017.

[5]. Konstantinos Menychtas, Kai Shen, Michael L. Scott. Enabling OS Research by Inferring Interactions in the Black-Box GPU Stack. In Proceedings of the 2013 USENIX conference on Annual Technical Conference, 2013, pp. 291-296. Available at: https://www.usenix.org/system/files/ conference/atc13/atc13-menychtas.pdf, accessed accessed 12.06.18

[6]. Unified EFI, Inc. Platform Initialization (PI) Specification. Version 1.6. 2017. Available at: http://www.uefi.org/sites/default/files/resources/PI Spec 1 6.pdf, accessed 09.06.18

[7]. Jeff Muizelaar, Pekka Paalanen. In-kernel memory-mapped I/O tracing Available at: https://www.kernel.org/doc/Documentation/trace/mmiotrace.txt, accessed 12.06.18

[8]. Arm Holdings. ARM1176JZ-S Technical Reference Manual. Available at: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0333h/ Caceaije.html, accessed 12.06.18

[9]. NXP Semiconductors. e500mc Core Reference Manual. Available at: http://cache. freescale.com/files/32bit/doc/ref manual/E500MCRM.pdf, accessed 09.06.18

[10]. Intel. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. Available at: http://www.ece.cmu.edu/_ece845/sp17/ docs/vt-overview-itj06.pdf, accessed 12.06.18

[11]. BeaEngine. Length Disassembler Engine for Intel 64-bit processors. Available at: https://github.com/BeaEngine/lde64, accessed 12.06.18

[12]. CupertinoNet. EfiPkg, AppleSmcIo protocol. Available at: https://github.com/ CupertinoNet/EfiPkg, accessed 12.06.18

[13]. Crowdstrike. Alex Ionescu. "Spell"unking in Apple SMC Land. 2013. Available at: http://www.nosuchcon.org/talks/2013/D1 02 Alex Ninjas and Harry Potter.pdf, accessed 09.06.18

Эмуляция ввода-вывода оборудования с отображением в ОЗУ внутри ядер операционных систем

[1,4] *В. Ю. Чепцов <cheptsov@ispras.ru>*
[1,2,3,4] *А. В. Хорошилов <khoroshilov@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25;*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1,*
[3]*Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[4] *Высшая школа экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20.*

Аннотация. Необходимость эмуляции оборудования часто возникает на различных стадиях цикла разработки, миграции оборудования или обратной разработки. Реализация алгоритмов, связанных с конкретным устройством, сама по себе является нетривиальной задачей, но интеграция эмулятора с существующей средой, например, драйверами, предназначенными для работы с реальным оборудованием, зачастую оказывается не менее сложной. Устройства, полагающиеся на ввод-вывод с отображением в оперативную память, представляют особый интерес, так как в этих случаях, в отличие от использования портов ввода-вывода, гораздо меньше вероятность, что целевая платформа предоставит интерфейс для перехвата операций. Один из распространённых подходов, широко используемый в ПО виртуальных машин, состоит в том, чтобы поместить всю операционную систему под гипервизор и создать внешний эмулятор. Однако это может быть нежелательно по причинам сложности гипервизора, потери производительности, дополнительных требований к аппаратному обеспечению и пр. В данной статье такой подход распространяется на ядро, и предлагается описание возможности построить эмулятор, прибегая лишь к существующим интерфейсам, предоставляемым операционной системой. Ввиду частой доступности MMU и механизмов защиты страниц, позволяющих перехватывать доступ записи и чтения, предполагается, что предлагаемый подход может быть использован на значительном количестве целевых платформ. В статье приводится подробное рассмотрение проблем, возникающих при написании конкретной реализации, и приводятся способы её упрощения и оптимизации в зависимости от возможностей целевой платформы, эмулируемого протокола и иных требований к задаче. В качестве экспериментального доказательства работоспособности предлагаемого подхода приводится реализация эмулятора SMC для платформы x86.

Ключевые слова: эмуляция оборудования; ввод-вывод с отображением в ОЗУ; модули ядра

DOI: 10.15514/ISPRAS-2018-30(3)-9

Для цитирования: Чепцов В.Ю., Хорошилов А.В. Эмуляция ввода-вывода оборудования с отображением в ОЗУ внутри ядер операционных систем. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 121-134 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-9

Список литературы

[1]. Jeremy Sugerman, Ganesh Venkitachalam, Beng-Hong Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In Proceedings of the General Track: 2001 USENIX Annual Technical Conference, 2001, pp. 1-14. Режим доступа: http://static.usenix.org/legacy/publications/library/proceedings/usenix01/sugerman/sugerman.ps, дата обращения 12.06.18

[2]. Keith Adams, Ole Agesen. A Comparison of Software and Hardware Techniques for x86 Virtualization. In Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, 2006, pp. 2-13. Режим доступа: https://www.vmware. com/pdf/asplos235 adams.pdf, дата обращения 09.06.18

[3]. Yusuke Suzuki, Shinpei Kato, Hiroshi Yamada, and Kenji. GPUvm: Why Not Virtualizing GPUs at the Hypervisor? In Proceedings of the 2014 USENIX Annual Technical Conference, 2014, pp. 109-120. Режим доступа: https://www.usenix.org/system/files/conference/ atc14/atc14-paper-suzuki.pdf, дата обращения 12.06.18

[4]. Hangchen Yu, Christopher J. Rossbach. Full Virtualization for GPUs Reconsidered. In Proceedings of the Annual Workshop on Duplicating, Deconstructing, and Debunking, 2017.

[5]. Konstantinos Menychtas, Kai Shen, Michael L. Scott. Enabling OS Research by Inferring Interactions in the Black-Box GPU Stack. In Proceedings of the 2013 USENIX conference on Annual Technical Conference, 2013, pp. 291-296. Режим доступа: https://www.usenix.org/system/files/ conference/atc13/atc13-menychtas.pdf, дата обращения 12.06.18

[6]. Unified EFI, Inc. Platform Initialization (PI) Specification. Version 1.6. 2017. Режим доступа: http://www.uefi.org/sites/default/files/resources/PI Spec 1 6.pdf, дата обращения 09.06.18

[7]. Jeff Muizelaar, Pekka Paalanen. In-kernel memory-mapped I/O tracing Режим доступа: https://www.kernel.org/doc/Documentation/trace/mmiotrace.txt, дата обращения 12.06.18

[8]. Arm Holdings. ARM1176JZ-S Technical Reference Manual. Режим доступа: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0333h/ Caceaije.html, дата обращения 12.06.18

[9]. NXP Semiconductors. e500mc Core Reference Manual. Режим доступа: http://cache. freescale.com/files/32bit/doc/ref manual/E500MCRM.pdf, дата обращения 09.06.18

[10]. Intel. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. Режим доступа: http://www.ece.cmu.edu/_ece845/sp17/ docs/vt-overview-itj06.pdf, дата обращения 12.06.18

[11]. BeaEngine. Length Disassembler Engine for Intel 64-bit processors. Режим доступа: https://github.com/BeaEngine/lde64, дата обращения 12.06.18

[12]. CupertinoNet. EfiPkg, AppleSmcIo protocol. Режим доступа: https://github.com/ CupertinoNet/EfiPkg, дата обращения 12.06.18

[13]. Crowdstrike. Alex Ionescu. "Spell"unking in Apple SMC Land. 2013. Режим доступа: http://www.nosuchcon.org/talks/2013/D1 02 Alex Ninjas and Harry Potter.pdf, дата обращения 09.06.18

Building Modular Real-time software from Unified Component Model

1,2 K.A. Mallachiev <mallachiev@ispras.ru>
1,2,3,4 A.V. Khoroshilov <khoroshilov@ispras.ru>
[1] Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
[2] Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[3] Moscow Institute of Physics and Technology,
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4] Higher School of Economics.
20, Myasnitskaya Ulitsa, Moscow 101000, Russia

Abstract. Modern real-time operating systems are complex embedded product made by many vendors: OS vendor, board support package vendor, device driver developers, etc. These operating systems are designed to run on different hardware; the hardware often has limited memory. Embedded OS contains many features and drivers to support different hardware. Most of the drivers are not needed for correct OS execution on a specific board. OS is statically configured to select drivers and features for each board. Modularity of OS simplifies both configuration and development. Splitting OS to isolated modules with well-specified interfaces reduces developers' needs to interact during joint development. The configurator, in turn, can easily compose isolated components without component developers. We use formal models to specify components and their composition. Formal model describes the behavior of components and their interaction. Usage of formal models has many benefits. Models contain enough information to generate source code in C language. Our model is executable; this allows configurator to quickly verify the correctness of component configurations. Moreover, model contains constraints on its parameters. These constraints are internal consistency or some external properties. Constraints are translated into asserts in generated source code. Therefore, we can check these constraints both at model simulation and at source code execution. This paper presents our approach to describe such models at Scala language. We successfully tested the approach in RTOS JetOS.

Keywords: components; modularity; RTOS; formal models; code generation

DOI: 10.15514/ISPRAS-2018-30(3)-10

For citation: Mallachiev K.A., Khoroshilov A.V. Building Modular Real-time software from Unified Component Model. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 135-148. DOI: 10.15514/ISPRAS-2018-30(3)-10

1. Introduction

Modern embedded operating systems support several CPU architectures and a lot of peripheral devices. OS contains many drivers to support numerous different hardware. Embedded OS are often designed for execution in a restricted environment, for example, with limited memory. Most of the drivers are not needed for correct OS execution on some specific board and spend valuable resources. Therefore, OS must support configuration to select drivers, which will execute on the target hardware.

Static OS configuration is used in cases when it is known in advance, on which hardware the OS image is going to be executed. Static means that configuration is performed on the host machine before OS loading to the target machine. The result of static OS configuration is the final image, which can be run on the target. Static configuration allows keeping final image small.

Typically, there are two roles taking part in the process of OS image building. The first role is a developer of whole OS or some driver. Developer implements his part in some programming language, writes documentation and provides support of source code and documentation. The second one is a system integrator who is responsible for correct OS configuration for specific task of specific board. Usually the system integrator does not change OS source code.

Besides simple selecting, which driver will be in the final OS image, many operating systems support finer tuning. For example, configuration allows selecting file system for each hard drive, or set IP address that will be used by network stack. These details are configured statically because for embedded OS and especially for safety-critical systems simplicity is more important than generality.

It is a natural desire to divide the operating system into isolated components, but not every part of the OS can be isolated. For example, OS core often is strongly coupled and might be divided into isolated components only if the core will be fully redesigned to support new architecture.

If we investigate configurations of the same OS on different boards, then we will see that there is the most variable part in the OS. We call this part *OS drivers*. OS drivers contain device drivers and some services such as network stack, file system, logging, etc. Our work aimed to support flexible configuration of OS drivers.

It is common that there are many vendors involved in building of OS drivers. When services or drivers are strongly coupled, their developers have to interact a lot. Therefore, splitting OS drivers into independent isolated components helps to simplify and accelerate development.

Component should interact with each other. Appearance of fixed interface between components would make component development easier. Moreover, fixed interface can make system flexible. Only connected components can interact, and only component with the same interfaces can be connected. System integrator is responsible for connection of the components.

Suppose that system integrator created a composition of the components, which describes how each component is configured and how components are connected. We call component-based system flexible if the system integrator can:

- modify configuration of the single component without modifying others,
- substitute component with another one of the same interface without modifying other components,
- add a new component between two other connected components without modifying any component configuration except the new one.
- add to composition a copy of existing component, and they should not disturb each other.

We are developing an embedded real-time operating system for civil aircraft computers called JetOS [1]. JetOS is ARINC-653 compliant and statically configured. Approaches presented in this paper are designed for JetOS. Since JetOS is a RTOS, we are focused on minimizing the overhead added by component-based system.

2. Related Works

Classical distributed component models like Enterprise JavaBeans, CORBA and CORBA Component Model [2, 3] define components and interfaces between them. These models allow substituting one component with another one if both have the same interfaces. Brokers dynamically change components configuration. This dynamic configuration is not suitable for embedded systems with static configuration.

Ideas to separate OS appeared long ago in microkernels. Microkernel architecture's [4, 5] primary goal is to separates OS into independent servers that could be isolated from each other. Servers interact through inter-process communication (IPC). IPC calls are typed and servers with the same interface can substitute one another. But there cannot be two servers with the same interface; therefore, this model is not suitable for our tasks too.

OS-Kit [6] and eCos [7] apply modularity benefits into OS development process. They provide a set of OS components, which are used as building blocks to configure an OS. For configuration, eCos uses the Component Definition Language (CDL), an extension of the existing Tool Command Language (Tcl) scripting language. Configuration is represented as feature tree with internal dependencies, group and feature constraints. Enabling of one component can lead to enable of whole components subtree. Components can have calculated value in configuration, which are calculated based on other configuration parameters. However, this is not enough for our task. Configurator cannot manage component connections and cannot add copies of the same component.

µC/OS-II kernel uses THINK component framework [8, 9]. THINK is an implementation of the FRACTAL component model that aims to take into account the specific constraints of embedded systems development. Component describes through its interface. Interaction between components is possible after establishment

of *bindings* between their interfaces. Binding is a communication channel between two or more components. Binding can be created between components of a distributed system (RPC binding). This concept also does not allow having several copies of the same component in the composition.

VxWorks is a popular embedded operating system. VxWorks board support package (BSP) is divided into components. Components interfaces are declared in Component Description Language (CDL). Note that this CDL is different from the CDL used in eCos. BSP developer can construct BSP from existing component and can add their own components. However, this system is not flexible. For example, each component has fixed list of component names, with which it can interact.

We are not aware of any component-based model with the following set of features:

- static configuration;
- low overhead;
- flexible configuration (in all aspects described in the introduction);
- type checking of the connection, i.e. checking that connected components have the same interface.

3. Component-based Model

Our model is component-based. Component has state, which is changed during model execution, and configuration, which is immutable. Components can communicate with other components via ports. Port is a set of functions; there are two kinds of ports: input ports and output ports. Output port can be connected with input port. Set of port function signatures is called port type. Only input and output port of the same port type can be connected.

Each function of a component input port has an assigned handler inside the component. Call of output port function leads to the call of connected input port, which, in turn, calls the assigned handler. These calls are standard function call, or in other words synchronous call inside the same thread. Therefore, component loses control during output port call.

Thus, port call keeps the current thread. Threads cannot be created dynamically during model execution. Threads count is constant during execution.

If component needs an additional thread, then this should be explicitly specified in the model. These components are called *active*. Active components have special handlers, which are called periodically or once in the context of the new thread. We call these handlers the *activity handlers*.

In order to facilitate component reuse we introduce the concepts of a *component type* and a *component instance*. Each component type can have any number of instances. The components described above are close to component instances.

Component type contains types of component state and configuration, but not their values. Component type contains types and names of input and output ports, but not their connection. In addition, component type contains implementation of:

- component initialization function, which is called at start and is used to initialize state based on the configuration;
- handlers assigned with input ports, if component has any;
- activity handlers if component is active.

Instances have unique values of state and configuration. It is easy to see that concepts of component type and component instance are similar to terms “class” and “class object” respectively.

3.1 Component Developer View

Component developer designs component state structure, how it should be initialized base on configuration and how it is changed during execution. Developer chooses types of configuration parameters. Developer does not aware of specific configuration parameters values, but he can add constraints on the values. He designs component input and output ports and implements handlers for input ports. Component’s input and output ports restrict component developer’s knowledge about “outside world”. He does not know how many instances of his component will be created or how they will be connected.

Component developer’s definition of component types consists of two parts: component type specification and implementation. Specification contains:

- component type name
- component input and output port names and their types
- structure of component configuration
- component’s purpose description: how it should be configured and in which environment its input ports should be called.

The rest of the information is private for component and is considered as implementation part.

3.2 System integrator view

System integrator gets specification of all component types in the system. System integrator decides how many instances of each component should be created and how they should be connected for solution of the specific problem. For each instance, integrator sets its configuration values.

3.2 Simple example

Suppose that component developer created *Amplifier* component type. *Amplifier* has single input port “in” and single output port “out”. In addition, it has single configuration parameter “factor”. Components aim is to amplify input signal from “in” port by factor “factor” and put output to “out” port.

Suppose that the system integrator wants to pass signal from two sensors to a single actuator, but he should amplify signal from first sensor by factor of 2 and from

second one by factor of 10. System integrator decides to use *Amplifier* component type. He does not worry about implementation, only interfaces matters to him. For simplicity, let us assume that all ports have the same type. Amplifier component type as seen by system integrator can be seen at Fig. 1

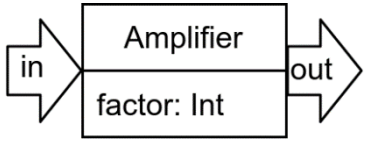


Fig. 1. Graphical representation of Amplifier component type specification

System integrator creates two instances of *Amplifier* component type: “amp1” with configuration value “factor” equal to 2 and “amp2” with configuration value “factor” equal to 10. Then connects them accordingly to sensors and to actuator. Scheme of the result can be seen at Fig. 2

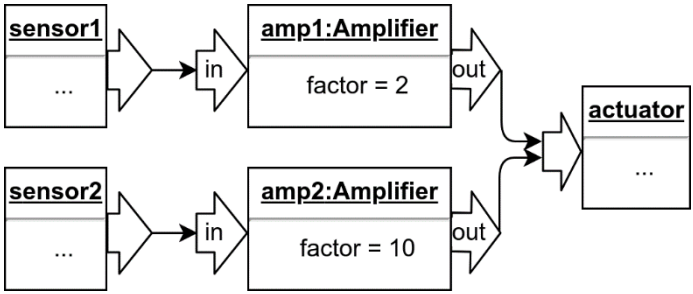


Fig. 2. Amplifier instances connection scheme

4. Prototype

In previous work [10], we implemented component-based approach in C language with some YAML code. We used common approach to apply object-oriented ideas in C language. Component state and configuration is presented as C structure, which explicitly passed to all component functions. Wrappers hid calls to output ports.

There was a lot of boilerplate code used to create component instances, describe their configuration, and their connections, in component type specification and its wrappers implementation.

To reduce amount of handwork we started to use YAML — simple declarative language. In the YAML developer specifies component type state, configuration, input and output ports, names of functions-handler for input ports. System integrator describes in the YAML component instances, their configuration and connections. We generated C code based on these YAML specifications.

This approach has some disadvantages.

- Component developer has to manually keep consistent two files (in YAML and C languages). Change in one file leads to change in another one.

- Component developer's workflow is not comfortable: after change in YAML code generation should be processed and only then C code should be updated accordingly.
- System integrator can connect instances incorrectly (this does not apply to type checking, which is performed during compilation) and cannot see the problem until final OS image is prepared and executed in target hardware.

5. Model-Based approach

We decided to go further along the path of abstraction and use abstract models of components and their composition. We use formal executable models. This has many benefits. Model contains more information than source code, thus source code can be generated based on the model. In addition, executable model allows simulating instances behaviour and their interaction. This is very useful for system integrator to quickly verify the correctness of configurations. Moreover, formal model can be used to formally verify its internal consistency.

We use Scala language to model components. Scala is a functional object-oriented language that suits us well.

5.1 Model Description

5.1.1 Component Developer View

Component type is presented as Scala class inherited from interface (trait) «Component». Component configuration and state are the class fields with fixed names «config» and «state» respectively.

Active components have functions, which are called periodically or once. If component type inherits trait «RunOnce» then it should implement function «start», which will be called once after component initialization. If component type inherits class «Periodically», then it should implement function «periodically»; the frequency of the call is determined by the configuration.

For example, consider "Counter" component type, at Fig. 3, which has a state but no configuration. State contains value «callCount», which is initialized with zero. Function «periodically» increases «callCount» on every call.

```
class Counter extends Periodic with Component {
  class State(val callCount: Int)
  var state = new State(0)

  type Config = Unit
  val config = ()

  def periodically = {callCount += 1}
}
```

Fig. 3. «Counter» component type

Port types are declared as interfaces (traits). Input ports are defined inside component type class as objects, which inherited port type. Output port are class fields with type of port type. Output ports values are passed as component type constructor parameters. It is worth noting that output ports can be passed by name to constructor, this allows initializing component instances with cycle connections among them.

Example of input/output ports for “Amplifier” component type (defined in previous sections) can be seen at Fig. 4 Model can have constraints on state and configuration parameters values. These constraints are defined using Scala require function. Example of require statement for “Amplifier” component type can be seen at Fig. 5.

```
trait SignalProcessor {
  def processSignal(s:Int): Int
}

class Amplifier(out: =>SignalProcessor)...{
  ...
  object in extends SignalProcessor {
    def processSignal(s: Int): Int = {
      val processed = process(s)
      out.processSignal(processed)
    }
  }
}
```

Fig. 4. Port type SignalProcessor and ports of «Amplifier» component type. The component type has input port «in» and output port «out», both of them have type SignalProcessor. here is an implementation of function processSignal of «in» port. Port «out» passed-by-name. Scala syntax may be confusing, here function processSignal returns result of out port call

```
class Amplifier...{
  class Config(val factor: Int) {
    require (factor>0 && factor<50)
  } ...
}
```

Fig. 5. Configuration constraint for «Amplifier» component type; «factor» can take values only in the interval from 1 to 49

5.1.2 System Integrator View

System integrator creates instances of component type and connects them. For each instance, he defines its configuration parameters values.

As an example of component instances and their connections, consider model of the scheme depicted in the Fig. 2. This model can be seen at Fig 6.

```
val actuator = new Actuator

val amp1 = new Amplifier(actuator.in) {
  val config = new Config(factor = 2)
}
sensor1 = new Sensor(amp1.in)

val amp2 = new Amplifier(actuator.in) {
  val config = new Config(factor = 10)
}
sensor2 = new Sensor(amp2.in)
```

Fig. 6. «Amplifier» instances connection scheme

5.1.3 Preconfigured components

There is often component which have configuration parameters that have the same value in different configuration. To simplify configuration process for system integrator, we can define new component type, in which these parameters are fixed and cannot be configured. New component type class constructor calls constructor of the original one with values of these parameters. For example, it is possible to define "AmplifierBy2" which amplifies signal by fixed factor of 2.

It is more interesting to define new component, which is a composition of existing components. This is useful if some compositions are used often. Our approach assumes unified modeling of components and their composition. This allows using component-composition transparently for system integrator.

As an example, assume that there are component type «Amplifier» and «Filter», that are often connected. We create a new component type «AmplifyAndFilter» that is the composition of «Amplifier» and «Filter» Graphical representation of the «AmplifyAndFilter» component type can be seen at Fig. 7 and implementation at Fig. 8.

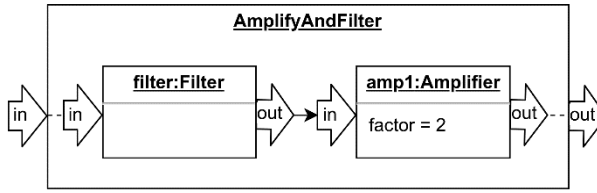


Fig. 7. Graphical representation of «AmplifyAndFilter» component type.

```
class AmplifyAndFilter(out: SignalProcessor)
    extends Component {
  val amp = new Amplifier(out) {
    val config = new Config(factor)
  }
  val filter = new Filter(amp.in)

  object in extends SignalProcessor {
    def processSignal(x:Int):Int = filter.in(x)
  }
}
```

Fig. 8. Implementation of «AmplifyAndFilter» component type

5.2 Model Usage

We use model to simulate instances behaviour and their interaction. We can verify that constraints are hold during simulation. In addition, we can write tests (unit and integration) to check that component model is correct.

We use model to generate C code, which gets into JetOS. We statically parse Scala code, extract needed information and translate it into C code.

Generated C code structurally looks much like code generated by prototype based on YAML files. We use same approach to model OOP in C language.

Some parts of the model can be translated into C without modifications, for example, simple operations and function calls. Some parts modified automatically during translation, but some can not be automatically translated without human help.

JetOS has strict coding style and, for instance, function can not have more than one `return` statement. We can generate code according this code style and, for example, we can automatically substitute several return statements in the model with a single one in the generated code.

As was mentioned, there are also statements, which cannot be easily translated into C. In addition, there are situations when generator tool cannot get enough information statically analysing Scala code. To solve these problems we add annotations to Scala code. Annotations does not change behaviour of model, they used only to provide additional information for the generator tool.

We use annotations to highlight input and output ports and their type interfaces. Annotations are «inport», «outport» and «interface» for input ports, output ports and port types respectively. As an example, Fig, 9 shows «Amplifier» component type with annotations.

```
@interface
trait SignalProcessor {
  def processSignal(s:Int): Int
}

class Amplifier(@outport out: SignalProcessor)...{
  ...
  @inport
  object in extends SignalProcessor {
    def processSignal(s: Int): Int = {
      val processed = process(s)
      out.processSignal(processed)
    }
  }
}
```

Fig. 9. Port type SignalProcessor and ports of «Amplifier» component type with annotations.

Scala language has rich syntax and not every statement can be easily translated to C. We allow annotating blocks of Scala code or Scala functions with C code. Fig. 10 contains partial example.

```
@C_code(code="int process(int* array) {...}")
def process(lst:List[Int]) = {...}
```

Fig. 10. C_code annotation example

This C_code annotation allows iteratively develop generator tool. At start, when tool supports only a few Scala statements, almost all code has C annotations. When support for new Scala statements adds to the tool, C annotations for these statements are no longer needed. Therefore, during tool development number of C_code annotations decreases.

6 Future Work

First, we still do not support many Scala statements and have a lot of C_code in our models. We are going to fix this in the new versions of generator tool.

For now, system developer should write Scala code by hand. This Scala code is very simple and matches a simple pattern. Thus, we can generate this Scala code from some GUI interface. Configuration constraints of the model can be extracted and added to this tool. This is one of optional future works.

Furthermore, formal model is a powerful tool and allows much more than C code generation. Formal model can be used for model checking and formal verifying internal consistency, preconditions or state invariants.

Tests and requirements can be generated based on the model and requirement generation is our next task. Requirement is the most important part of safety-critical system certification. Requirement writing is a hard handwork and automation (at least partial) will be very helpful.

7 Conclusion

The paper presents continuation of the work on modularity of RTOS. OS drivers are decomposed into isolated components. System integrator carries out component composition, and it can be done without contacting component developers and without writing C code.

We use a unified formal model to specify both components and their composition. Model, which is written in Scala language, is used to generated C code.

Also, model is executable, this allows system integrator to quickly verify correctness of composition. Model contains constraints on the model parameters. These constraints are tested during model simulation, also constraints can be translated into asserts in the generated C code.

Model-based approach still has disadvantage since the model is divided in two parts written in two languages, which have to be manually kept consistent. However, C code for some Scala statement is placed right before the statement, we hope that this will stimulate developers to update parts synchronously. Maturing of the generator tool decreases amount of C code in the model and reduces the importance of the problem.

The approach has been successfully tested on OS drivers of JetOS — ARINC-653 compliant RTOS. ARINC-653 has restrictions on the code executed in OS. For instance, resources (like buffers, semaphores, threads, etc.) can be requested only during initialization stage. Model restriction on threads creation apply well to ARINC-653 restrictions. Moreover, constructor code of the component type class is executed during initialization stage. Thus, component can request resources in the constructor.

References

[1]. K.M. Mallachiev, N.V. Pakulin, and A.V. Khoroshilov. Design and architecture of real-time operating system. Trudy ISP RAN / Proc. ISP RAS, vol. 28, no. 2, 2016, pp. 181–192. DOI: 10.15514/ISPRAS-2016-28(2)-12

[2]. J. Siegel and D. Frantz. CORBA 3 fundamentals and programming. John Wiley & Sons New York, NY, USA, 2000, vol. 2.

[3]. N. Wang, D. C. Schmidt, and C. O'Ryan. Overview of the corba component model. In Component-Based Software Engineering. Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 557–571.

[4]. A. Gefflaut, T. Jaeger, Y. Park, J. Liedtke, K. J. Elphinstone, V. Uhlig, J. E. Tidswell, L. Deller, and L. Reuther. The sawmill multiserver approach. In Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, 2000, pp. 109–114.

[5]. I. Boule, M. Gien, and M. Guillemont. Chorus distributed operating systems. Computing Systems, vol. 1, no. 4, 1988, pp. 305-370.

[6]. B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The flux oskit: A substrate for kernel and language research. ACM SIGOPS Operating Systems Review, vol. 31, no. 5, 1997, pp. 38–51.

[7]. A. Massa, Embedded software development with eCos. Prentice Hall Professional Technical Reference, 2002.
[8]. J.-P. Fassino, J.-B. Stefani, J. L. Lawall, and G. Muller. Think: A software framework for component-based operating system kernels. In Proceedings of the USENIX Annual Technical Conference, General Track, 2002, pp. 73–86.
[9]. F. Loiret, J. Navas, J.-P. Babau, and O. Lobry. Component-based real-time operating system for embedded applications. In Proceedings of the International Symposium on Component-Based Software Engineering. Springer, 2009, pp. 209–226.
[10]. K. Mallachiev, N. Pakulin, A. Khoroshilov, and D. Buzdalov. Using modularization in embedded OS. Trudy ISP RAN / Proc. ISP RAS, vol. 29, issue. 4, 2017, pp. 283–294. DOI: 10.15514/ISPRAS-2017-29(4)-19

Построение модульного программного обеспечения на основе однородной компонентой модели

[1,2] *К.А. Маллачиев <mallachiev@ispras.ru>*
[1,2,3,4] *А.В. Хорошилов <khoroshilov@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25,*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[3] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[4] *Высшая школа экономики, 101000, Россия, г. Москва, ул. Мясницкая, д. 20*

Аннотация. Современные операционные системы реального времени являются сложным продуктом, разрабатываемым многими поставщиками: непосредственными разработчиками ОС, поставщиками пакета поддержки аппаратуры, разработчиками драйверов устройств и т.д. Такие ОС спроектированы так, чтобы иметь возможность запускаться на различном оборудовании, часто имеющем ограниченные ресурсы. Встраиваемые ОС содержат множество настроек и драйверов для поддержки разной аппаратуры. Большинство из этих драйверов являются излишними для запуска ОС на каком-то конкретном оборудовании. ОС статически конфигурируется для выбора набора драйверов и настроек для каждого типа аппаратуры. Модульность ОС упрощает как разработку ОС, так и ее конфигурирование. Разделение ОС на изолированные модули с фиксированными интерфейсами уменьшает необходимость взаимодействия между разработчиками в ходе совместной разработки. Мы используем формальные модели для описания компонентов и их взаимодействия. Использование формальных моделей приносит большую пользу. Описываемые модели содержат достаточно информации для генерации исходного кода компонента на языке Си. Предоставляемые модели являются исполняемыми, что позволяет человеку, отвечающему за конфигурацию, быстро проверить правильность заданной конфигурации. Кроме того, модель содержит ограничения на конфигурационные параметры. Примером таких ограничений являются ограничения на внутреннюю согласованность модели. При генерации исходного кода такие ограничения транслируются в специальные проверки

на уровне исходного кода. Следовательно, ограничениями могут быть проверены как во время симуляции модели, так и во время исполнения исходного кода. В данной работе представлен подход к описанию таких моделей на языке программирования Scala. Мы успешно апробировали данный подход на основе ОС реального времени JetOS.

Ключевые слова: компоненты; модульность; ОСРВ; формальные модели; генерация кода

DOI: 10.15514/ISPRAS-2018-30(3)-10

Для цитирования: Маллачиев К.А., Хорошилов А.В Построение модульного программного обеспечения на основе однородной компонентой модели. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 135-148 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-10

Список литературы

[11]. K.M. Mallachiev, N.V. Pakulin, and A.V. Khoroshilov. Design and architecture of real-time operating system. Trudy ISP RAN / Proc. ISP RAS, vol. 28, no. 2, 2016, pp. 181–192. DOI: 10.15514/ISPRAS-2016-28(2)-12

[1]. J. Siegel and D. Frantz. CORBA 3 fundamentals and programming. John Wiley & Sons New York, NY, USA, 2000, vol. 2.

[2]. N. Wang, D. C. Schmidt, and C. O'Ryan. Overview of the corba component model. In Component-Based Software Engineering. Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 557–571.

[3]. A. Gefflaut, T. Jaeger, Y. Park, J. Liedtke, K. J. Elphinstone, V. Uhlig, J. E. Tidswell, L. Deller, and L. Reuther. The sawmill multiserver approach. In Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system, 2000, pp. 109–114.

[4]. I. Boule, M. Gien, and M. Guillemont. Chorus distributed operating systems. Computing Systems, vol. 1, no. 4, 1988, pp. 305-370.

[5]. B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The flux oskit: A substrate for kernel and language research. ACM SIGOPS Operating Systems Review, vol. 31, no. 5, 1997, pp. 38–51.

[6]. A. Massa, Embedded software development with eCos. Prentice Hall Professional Technical Reference, 2002.

[7]. J.-P. Fassino, J.-B. Stefani, J. L. Lawall, and G. Muller. Think: A software framework for component-based operating system kernels. In Proceedings of the USENIX Annual Technical Conference, General Track, 2002, pp. 73–86.

[8]. F. Loiret, J. Navas, J.-P. Babau, and O. Lobry. Component-based real-time operating system for embedded applications. In Proceedings of the International Symposium on Component-Based Software Engineering. Springer, 2009, pp. 209–226.

[9]. K. Mallachiev, N. Pakulin, A. Khoroshilov, and D. Buzdalov. Using modularization in embedded OS. Trudy ISP RAN / Proc. ISP RAS, vol. 29, issue. 4, 2017, pp. 283–294. DOI: 10.15514/ISPRAS-2017-29(4)-19

Methods of protecting decentralized autonomous organizations from crashes and attacks

A.A. Andryukhin <Alexandr@kcdigital.ru>
KCD, office 3, 131, prospect Mira, Moscow, 129226, Russia

Abstract. Field of study: Blockchain technology, decentralized autonomous organizations, smart contract and their resistance to attacks and failures. Theoretical and practical significance: Due to the fact that such a form of organization is experimental, participants often face problems of attacks on the organization, the consequences of incorrectly written rules and of fraud. The task of creating decentralized autonomous organizations that are resistant to failures and attacks, and research on the causes of such problems has become relevant for software developers and architects. Goals and objectives of work: Investigation of attack algorithms and development of methods for ensuring the sustainability of decentralized autonomous organizations for attacks on the basis of analysis of the subprocesses of border events and logs using the methods of Process Mining. The methods to be developed should promptly identify and prevent inconsistencies between the alleged and actual behavior of smart contracts that lead to such errors in the operation, such as the content of spam contracts, empty transactions, increased block processing time, etc.

Keywords: blockchain; decentralized autonomous organizations; process mining; smart contract; security

DOI: 10.15514/ISPRAS-2018-30(3)-11

For citation: Andryukhin A.A. Methods of protecting decentralized autonomous organizations from crashes and attacks. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 149-164. DOI: 10.15514/ISPRAS-2018-30(3)-11

1. Introduction

1.1 Blockchain and crypto-currencies

In the past few years, thanks to the popularization of blockchain technology, which represents a continuous series of blocks containing information, built according to certain rules, there were created many services and applications using various crypto-currencies [13]. Many crypto-currencies are inextricably linked with this technology for such reasons: decisions on the blockchain do not require trust between the participants, they are open and validated. Success of the Bitcoin, decentralized crypto-currency with the capitalization of more than $ 10 billion, is of

genuine interest both in industry, government, and in science [2]. A whitepaper, written under the pseudonym Satoshi Nakamoto in 2008, is the basic document for any form of organization created on blocking technology. This document for the first time outlines the Bitcoin structure and introduces the concept of blockchain [30]. The theoretical basis used in the creation of decentralized autonomous organizations is based on the research of automatic verification systems [31, 32], cryptography [33, 34] and distributed databases [35, 36].

1.2 Decentralized autonomous organizations

The economic theory and research of organizations [19, 20], the theory of contracts [21, 22], auction mechanisms [23, 24], the theory of innovation [25, 26], as well as virtual organizations [27, 28] played an important role in the emergence of decentralized autonomous organizations [30].

Bitcoin can be called the very first decentralized autonomous organization created to carry out paid transactions [1]. The most famous decentralized autonomous organization, based on the Bitcoin code, was created in 2014 and was given the current name Dash in March 2015. Dash is currently experiencing a stage of rapid growth. In September 2017, the company's market capitalization was $ 2.5 billion.

However, the most promising platforms for the development of decentralized autonomous organizations are platforms that use smart contracts and the Turing-complete programming language, such as Ethereum [2].

On April 30, 2016, the first decentralized crowd-funding project, known as The Dao (Decentralized Autonomous Organization), was launched on the Ethereum blockchain platform. The organization was established as a venture capital fund with transparent and democratic flows of project financing, in which each investor would have a voice whose weight is directly proportional to the funds invested. The technology of smart contracts was laid for the first time in the basis of the functioning of the organization The DAO. The Dao in record time attracted more than $ 168 million in investment almost immediately became a target of intruders and was repeatedly attacked to steal or freeze funds. As a result of one of the attacks, more than $ 50 million was stolen from the organization, and as a result of the other, more than $ 150 million was frozen [14, 16]. The imperfection in the code of smart contracts and the existing vulnerabilities, as well as the inability to change them lead to so-called softfork and hardfork. The Dao is not the only decentralized organization deployed on Ethereum. Fermat (www.fermat.org), Digix.global also operate on the Ethereum blockchain platform and are managed collectively by the participants who own the tokens by voting.

1.3 Smart contracts

In 1994, cryptographer Nick Szabo proposed the use of cryptography and computer technology to automate the process of concluding, executing and auditing various contracts [29]. The development of this direction led to the creation of smart

contracts on the basis of the blockade - special electronic algorithms introduced into the block, where they are monitored by the decentralized computer network itself. This allows you to expand the capabilities of the block-up to a computing platform for centralized execution of common tasks [5].

Smart contracts allow you to exclude from the process of intermediaries because computer algorithm independently and automatically confirms the fulfillment of the terms of the contract and determines what to do with the asset for which the contract was created. Smart contracts are protected from uncoordinated changes in the terms of the transaction, allow you to automate the audit and make it in real time.

The most famous framework for smart contracts is Ethereum, a decentralized virtual machine, where the Turing-complete programming language is used to create smart contracts. A distinctive feature of Ethereum is the ability to transfer ETH crypto currency between users and contracts. Users create transactions on the Ethereum network in order to create a new contract, call a contract, or transfer ETH to a contract or another user. Blockchain allows you to track the status of each contract and the balance of each user.

Smart contracts are unchangeable: after they are deployed in the core network, updates and changes are not possible, they are publicly available. The main serious problem of creating smart contracts is their formal verification: for example, in the Etherium network, verifying the decentralized virtual machine (EVM) code is very difficult, so unverified smart contracts are often the subject of hacker attacks. Later in the article, known vulnerabilities and attacks will be examined using the example of the Ethereum network and the distributed decentralized autonomous organization The Dao.

In this article, special attention will be paid to the security of decentralized autonomous organizations, which are based on smart contracts, examines examples of existing attacks. The problem of attacks on DAO is currently relevant, although it is currently not very well covered [2, 4].

2. Structure of the DAO based on smart-contracts

A decentralized autonomous organization is a supposedly "democratic" organization operating in a distributed network through a combination of "smart" contracts and a rich scripting language. Technically, DAO is the implementation of a financial service by performing all necessary calculations directly in smart contracts when using the scripting language. A distributed ledger, for example, a host, provides a secure environment for computing and storing data across the entire network, and, as a consequence, eliminates the need for a central trusted party [1].

As an example of the structure of a decentralized autonomous organization, TheDAO can be considered, where the main smart contract is used, serving as a "factory" for sub-contracts, the number of which is already in the millions. Smart contracts in Ethereum run on Ethereum Virtual Machine (EVM), the predominant language of contracts is Solidity. A smart contract is an autonomous agent with its own software logic, an identification address in the network and the associated

balance of the Ether. After the initialization, the contract code can no longer be changed, the contract can be called repeatedly and stored on the network forever, until it executes the bytecode of the suicide instruction, after which the contract is no longer subject to a call and is called *dead* [7,9]. Each contract call is carried out by sending a transaction to the address of the contract together with the input data and charges (the so-called gas). Ideally, the entire mining network performs a function call and skips or does not miss the contract, depending on the consensus reached, based on the consensus protocol. The result of the calculation is replicated through the blockchain and provides a commission for the transaction for the miners in accordance with the established interest rates.

In addition to being used as a reward, the service fee also protects against *denial-of-service attacks* when an attacker tries to slow down the entire network by requesting time-consuming calculations. Each operation consumes a certain level of *gas*, the upper consumption threshold and the unit price of which are indicated in the transaction. Unused gas comes back, and if during the calculation all gas was consumed, then the process stops and all gas is lost.

EVM allows contract functions to have a local state, while contracts themselves can contain global variables stored in the blockchain. Contracts can also refer to other contracts via message calls. The output of these calls is part of the same transaction and is also returned during the runtime of the transaction. It is important to note that calls can also send the Ether to other contracts and non-contractual addresses. The balance of the contract can be read by any member of the blockchain, but it can be changed only by calls originating from other contracts or initiated from outside the transaction. Only contracts with white list addresses can receive funding from the organization, and track the addition of new contract addresses, the main purpose of which is financing, curators [9].

The main motive for the introduction of human control is the screening out of malicious addresses, through which the "51% attack" is carried out, the purpose of which is to transfer most of the company's funds to one block. After adding the contract address to the white list, further decisions on it are made by voting all the holders of the tokens. At the time of voting, the balance sheets of the voters are "frozen" to the voting results. The withdrawal of funds from the organization is possible only by creating a sub-organization, where the withdrawing funds is the sole curator. The decision on separation (creation of a new DAO) is also adopted by a general vote. The entire process of creating a new DAO takes a little more than 30 days [4, 10].

3. Vulnerabilities of DAO

Attacks of the DAO system based both on the technical imperfection of the system and on the behavioral characteristics of the DAO participants [15, 16, 17]. The behavior of participants allowed the appearance of the following types of attacks, some of which are still used for malicious actions in the system [4].

Stalker Attack. During the separation and creation of a subsidiary DAO in order to withdraw funds from the system (the withdrawal is possible only under this scheme), an attacker can seize tokens created by the DAO and have a negative impact on the withdrawal of funds.

Attack of the last moment. At the last moment of voting, a large investor is added with a hug number of tokens with which he votes "yes" and pushes an unprofitable or absurd project.

Attack of the value of the token. Sowing panic among tokens holders, forcing to sell tokens, and not invest in system projects. There is a buying up of tokens at a low price and the acquisition of a larger stake in the DAO.

Attack of extra-balance. The attacker provokes the separation of DAO to increase the book value of tokens. The more participants are separated from the DAO, the higher the value of the extra balance as a percentage.

Attack of 53%. Despite the huge amounts of 53% of DAO funds and curatorial verification of the addresses of financed contracts, there is a possibility of cartel collusion with the aim of raising funds for interrelated projects.

Attack of parallel voting. For the voting period, the balances of the voters are blocked, which can be used for voting for a malicious contract with a smaller voting period.

Attack on reward. To reduce the payments to the separated participants of the system, the remaining participants can deliberately create overheads for maintenance by looping money in fake contracts.

Logical vulnerability of voting. The nature of voting in the existing DAO does not allow to build a logical chain during voting. For example, (vote "yes" the proposal *A* if the proposal *B* is not funded). Because social processes are non-linear, it is impossible to foresee how competing or conflicting proposals run simultaneously.

Attacks that exploit the behavioral features of the system, for the most part, require tremendous resources and considerable training, while attacks based on technical vulnerabilities and bugs can be carried out with minimal costs, thus such attacks are the most interesting and dangerous.

According to the studies [5], the Ethereum blockchain contains over 34,000 vulnerable smart contracts per 1 million researched contracts. Vulnerable contracts were divided into 3 conditional groups: *suicidal contracts, prodigal contracts* and *greedy contracts* - such contracts allow either to block funds for an indefinite period, or to destroy the contract after implementation, or allow leakage means of purse to arbitrary users.

There are several types [2] of major vulnerabilities that make the contract dangerous for the system.

Call to the unknown. When the code is illiterate, the call, send, and delegate call primitives can result in sending to an unset address or returning a broadcast by calling a backup function.

Exception disorder. In Solidity, exceptions are used in the following cases:

- the gas has ended;

- the call stack has reached the limit;
- a command throw is executed [2].

However, in some cases (often these are chains of nested calls), exceptions can lead to an unplanned cancellation of the actions performed, while gas consumption is not returned [11].

Gasless send. The lack of gas in the transmission of Ether can cause unpredictable behavior.

Type casts. Using the compiler does not guarantee the correct operation of the contract.

Reentrancy. It can often be confusing to realize that if a function is not recursive, then it will not allow repeated repetitions. However, this misconception can lead to the fact that a non-recursive function starts a cycle of calls that ultimately consume all the gas [11].

Keeping secret. Fields in contracts can be both private and public for all users. However, declaring a field private does not guarantee its inaccessibility to others. This is due to the fact that to set the privacy of the field, the user must send the corresponding transaction to the miners who will then publish it in the blockchain. Since the blockchain itself is public, any user can check the contents of the transaction and make changes to the privacy of the field. In order to best protect the contract field, you need to use suitable cryptographic methods [12].

Immutable bugs. As already known, after the publication of the contract in the detachment, it is already impossible to change it, so contracts with errors can manifest themselves completely unpredictably. Sometimes, when the consequences of executing such contracts have an extremely negative impact on the entire detachment, the community comes to the decision to use *softfork* or *hardfork*.

Ether lost in transfer. Some addresses in the blockchain are not associated with either specific users or contracts, so when sending airtime to these addresses, it is lost irrevocably.

Stack size limit. The stack size is limited to 1024 frames. Every time there is a call to another contract or even yourself, the stack size increases by 1. If the rules for rejecting a call when reaching a stack limit are incorrectly set, then the attacker has the opportunity to exploit the vulnerability. The vulnerability was closed in 2016 by limiting gas at a rate of 63/64 from the existing one. Since the current gas limit is limited to 4.7M units, the depth of the stack is always less than 1024.

Unpredictable state. When sending a transaction to the network, the user can not always be sure of the status of the contract, which is determined by the cost of its fields and balance. This can happen because at the time of sending the contract status was changed by another transaction, or the contract contains dynamic variables associated with other contracts. Such vulnerability can be used by attackers to link the called contract to malicious components that allow stealing the broadcast.

Generating randomness. Due to the fact that execution of the bytecode on EVM is deterministic, i.e. all participants as a result of processing the transaction should

receive the same result, unless otherwise specified, to obtain non-deterministic results, some contracts (for example, games, lotteries) use pseudo-random number generators. Such blocks usually have timestamps. The vulnerability lies in the fact that an attacker can try to create his own block with the content controlled by him in order to evade the result of the generator and shift the probability of distribution of pseudo-random numbers.

Time constraints. Time constraints are used to identify permitted or mandatory actions and contain a timestamp that is consistent with all participants in the process. Contracts can extract timestamps and set their own. Attackers can exploit this vulnerability to gain temporary advantages over other participants in the process.

The Threat of Quantum Computing. One of the potential vulnerabilities is the instability of cryptography to quantum attacks. The most popular public-key encryption algorithms, for example, RSA in the near future can be destroyed with the help of a quantum computer.

4. Levels of attacks on smart-contracts

In connection with the fact that the basis of any decentralized autonomous organization is the implementation of smart contracts, the main attacks are aimed at them. The existing vulnerabilities of smart contracts can be conditionally divided into three classes, depending on the level at which the vulnerability is detected (Solidity, EVM bytecode, blockchain). Each vulnerability class can spawn one or more known attack types [2, 16, 17] (fig. 1).

In the study [2], the simplest test DAO was simulated,

```
contract SimpleDAO {
  mapping (address => uint) public credit;
  function donate(address to){credit[to] += msg.value;}
  function queryCredit(address to) returns (uint){
    return credit[to];
  }
  function withdraw(uint amount) {
    if (credit[msg.sender]>= amount) {
      msg.sender.call.value(amount)();
      credit[msg.sender]-=amount;
}}}
```

on which the following attacks, existing in real Ethereum, were made.

The DAO Attack. In the well-known attacks on the DAO, the purpose of which was to seize the organization's funds, the *call to the unknown* and *reentrancy* vulnerabilities were exploited, which could have a negative impact, because the broadcast was broadcast before the credit was reduced. Examples of contracts used in attacks:

```
contract Mallory {
  SimpleDAO public dao = SimpleDAO(0x354...);
  address owner;
  function Mallory(){owner = msg.sender; }
  function() { dao.withdraw(dao.queryCredit(this)); }
  function getJackpot(){ owner.send(this.balance); }
}
```

```
contract Mallory2 {
  SimpleDAO public dao = SimpleDAO(0x818EA...);
  address owner; bool performAttack = true;

  function Mallory2(){ owner = msg.sender; }

  function attack() {
    dao.donate.value(1)(this);
    dao.withdraw(1);
  }
```

```
  function() {
    if (performAttack) {
      performAttack = false;
      dao.withdraw(1);
  }}

  function getJackpot(){
    dao.withdraw(dao.balance);
    owner.send(this.balance);
}}
```

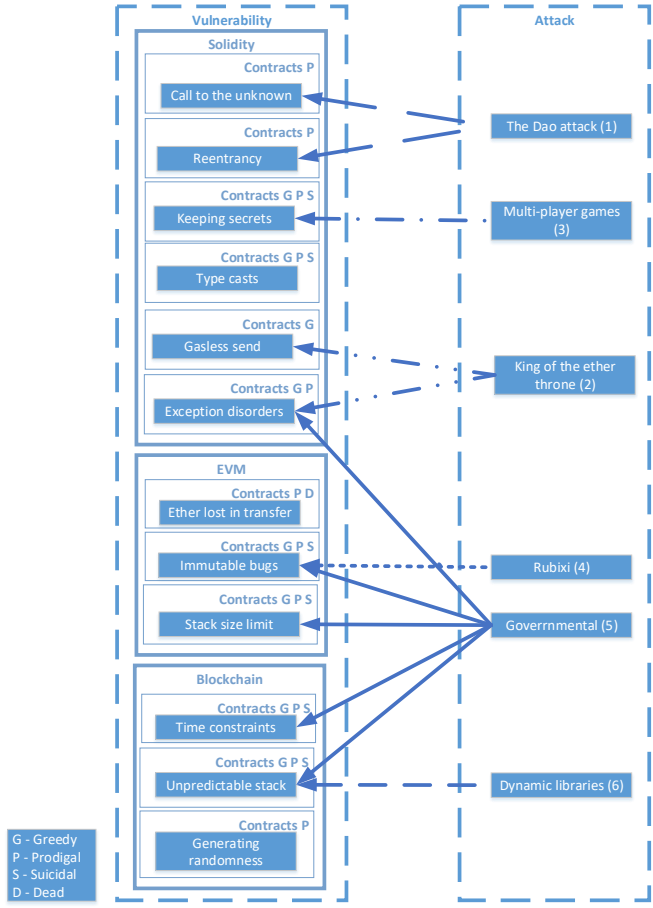


Fig. 1. Vulnerabilities of smart contracts

Attack in the game King of Ether throne. The game is represented by simplified contracts with vulnerabilities.

```
contract KotET {
  address public king;
  uint public claimPrice = 100;
  address owner;

  function KotET() {
    owner = msg.sender;  king = msg.sender;
  }

  function sweepCommission(uint amount)  {
    owner.send(amount);
  }

  function() {
    if (msg.value < claimPrice) throw;

    uint compensation = calculateCompensation();
    king.send(compensation);
    king = msg.sender;
    claimPrice = calculateNewPrice();
  }
  /* other functions below */
}
```

```
contract KotET {
  ...
  function() {
    if (msg.value < claimPrice) throw;
    uint compensation = calculateCompensation();
    if (!king.call.value(compensation)()) throw;
    king = msg.sender;
    claimPrice = calculateNewPrice();
}}
```

```
contract Mallory {

  function unseatKing(address a, uint w) {
    a.call.value(w);
  }

  function () {
    throw;
}}
```

At first glance, contracts seem fair, but the lack of a send return check (1) and intentional call exceptions (2) can result both in unfair winnings and in theft of contract funds after the game is over.

Attack in games with multiple players. In such games, hidden fields are often used, which are unknown during the game, but can be opened at the time of joining the game (vulnerability *keeping secrets*). An example of a similar game with existing vulnerabilities is represented by a contract

```
contract OddsAndEvens{
  struct Player { address addr; uint number;}
  Player[2] private players;
  uint8 tot = 0; address owner;

  function OddsAndEvens() {owner = msg.sender;}

  function play(uint number) {
    if (msg.value != 1 ether) throw;
    players[tot] = Player(msg.sender, number);
    tot++;
    if (tot==2) andTheWinnerIs();
  }

  function andTheWinnerIs() private {
    uint n = players[0].number
           + players[1].number;
    players[n%2].addr.send(1800 finney);
    delete players;
    tot=0;
  }

  function getProfit() {
    owner.send(this.balance);
  }
}
```

Using data from private fields, an attacker can lead a strategy of permanent winnings.

Attack of Rubixy. Was implemented in contracts that use the Ponzi scheme (financial pyramid). Attack was possible because the developers renamed the contract with DynamicPyramid Rubixy, forgetting to change the name of the constructor, which then became a function that everyone can call. Instead of a single use of DynamicPyramid when setting the owner's address, which is allowed to take profit, this function was used by intruders to set their addresses as owner addresses.

```
contract Rubixi {
  address private owner;
  function DynamicPyramid() { owner = msg.sender; }
  function collectAllFees() { owner.send(collectedFees); }
```

Attack GovernMental. As well as above, the contract is implemented by the Ponzi scheme. Money receives the final invested after 12 hours except for the fees of the organizers. After that, the array is cleared with the data of the participants. At some point, the list became so huge that it took much more gas to clean up the arrays than the maximum allowed for a single transaction.
A simplified version of the game with all the existing vulnerabilities looks like this

```
contract Governmental {
  address public owner;
  address public lastInvestor;
  uint public jackpot = 1 ether;
  uint public lastInvestmentTimestamp;
  uint public ONE_MINUTE = 1 minutes;

  function Governmental() {
    owner = msg.sender;
    if (msg.value<1 ether) throw;
  }

  function invest() {
    if (msg.value<jackpot/2) throw;
      lastInvestor = msg.sender;
      jackpot += msg.value/2;
      lastInvestmentTimestamp = block.timestamp;
  }

  function resetInvestment() {
    if (block.timestamp <
        lastInvestmentTimestamp+ONE_MINUTE)
      throw;

      lastInvestor.send(jackpot);
      owner.send(this.balance-1 ether);

      lastInvestor = 0;
      jackpot = 1 ether;
      lastInvestmentTimestamp = 0;
    }
  }
```

This scheme was also subjected to attacks using the *exception disorder* and *stack size limit* vulnerabilities. Thanks to these vulnerabilities, it became possible not to pay the winners to win by launching a new round of the game. Also, dishonest miners used the possibility of not adding new blocks to be the last ones invested, or adding a timestamp to the block in such a way that the block would become the last one each time.
An attack using dynamic libraries. Such attacks use the Unpredictable state vulnerability, since it is possible to update the library with malicious content after the publication of the contract.

5. Potential mitigations and solutions

Having considered the above vulnerabilities and attacks based on them, it is possible to draw conclusions and understand the need for steps to be taken in the field of DAO security.
Confidentiality. Many mistakenly accept conditional anonymity of transactions in the blockchain for real: transactions are recorded and stored in the public registry and are linked to the address of the account that does not contain information about the real person behind this account. However, identifying information can be

obtained through web trackers and cookies. In addition, the required data is often contained directly in smart contracts [17]. Lack of confidentiality is a serious threat when it comes to medical records, identity documents, credential management, and a number of closed financial documents. Strengthen confidentiality in several ways:

- addresses on the Diffie-Hellman-Merkle protocol on elliptical curves (ECDHM) will allow the use of the secret key by the two sides of the process;
- creation of a decentralized mix-protocol for joining a group of payments into one pool, with the possibility of tracking amounts in a private registry, without a third party;
- evidence with zero disclosure – the preservation of confidential information and at the same time the certification of its availability; it can be achieved by authentication of the "call-response" to verify the transaction, using the zkSNARK (zk-zero-knowledge, SNARK-Succinct Non-interactive Adaptive Argument of Knowledge) module for verification; it will make certain contract variables private, ensuring that they are stored out of the blockchain by users who using the SNARK protocol to prove that they adhere to its rules (requires a prior "trust"); the use of the zkSTARK (T-transparent, i.e. transparent) block is a simple protocol that relies solely on hashes and is better protected of quantum computers, because it does not use elliptic curves;
- use of obfuscation (code entanglement);
- use of oracles - parties transferring information between smart contracts and external data sources;
- use of the trusted environment for program execution.

Verification of smart contracts. The development of tools for verification, the introduction of verification formats will make sure that the smart contract behaves the way it was intended. The complexity of verification of smart contracts lies in the complexity of verification of the EVM bytecode. Verification of smart contracts will also reduce the risk of virus infection and hacker attacks. Verification guarantees greater accuracy, than traditional approaches, for example, testing [8].

Perfection of intra-organizational processes. Improving the voting processes by introducing a grace period that allows the movement of non-voting funds, adding the function of the voting office, prolonging the voting time for a statistical release, attracting more users to the process, developing secure withdrawal methods will prevent a number of attacks and increase trust in the system.

Improving the mechanism for achieving consensus. The use of the existing PoW (Proof-of-Work) protocol, which depends on computing power, jeopardizes the decentralization of the system and makes it possible for a cartel plot. Because Major mining pools have a great advantage over private miners in the extraction of blocks and profit distribution, centralization occurs in the blockage and several large mining pools own more than 70% of the hash. A more advanced PoS protocol

(Proof-of-Stake) practically excludes the possibility of bundle aggregations in terms of computing power, but at the same time it requires solutions to problems such as «Nothing at stake», when forming forecaster, the miners vote simultaneously for several different blocks on one altitude, or start fork from any place, getting validators of previous participants and creating a million blocks in a new blockchain, preventing users from understanding which of the blockchain is «correct».

Creation of the necessary tools for development.

At the moment, in the ecosystem of the toolkit of the developer of smart contracts, the weak points are:

- Integrated Development Environment (IDE);
- the code assembly system and compiler program;
- deployment tools;
- storage medium;
- debugging and logging tools;
- security audit;
- analytical tools.

Improving the development toolkit will have a positive impact on the functioning of the entire DAO.

References

[1]. Williams J. The Seconomics (Security-Economics) Vulnerabilities of Decentralized Autonomous Organizations. Lecture Notes in Computer Science, vol. 10476, 2017, pp. 171-179.

[2]. Atzei N., Bartoletti M., Cimoli T. A survey of attacks on Ethereum smart contracts (SoK). In Proc. of the International Conference on Principles of Security and Trust, 2017, pp. 164-186.

[3]. Mehar M. et al. Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack. Available at SSRN: https://ssrn.com/abstract=3014782, accessed 29.05.2018.

[4]. DuPont Q. Experiments in algorithmic governance: A history and ethnography of "The DAO," a failed decentralized autonomous organization. In Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance, Routledge, 2017, 212 p.

[5]. Nikolic I. et al. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. arXiv preprint arXiv:1802.06038, 2018.

[6]. Grossman S. et al. Online detection of effectively callback free objects with applications to smart contracts. Proceedings of the ACM on Programming Languages, vol. 2, issue POPL, article 48, 2017, 20 p.

[7]. Gurfinkel A. et al. The SeaHorn verification framework. In Proc. of the International Conference on Computer Aided Verification, 2015, pp. 343-361.

[8]. Bhargavan K. et al. Formal verification of smart contracts. In Proc. of the ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91-96.

[9]. Delmolino K. et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In Proc. of the International Conference on Financial Cryptography and Data Security, 2016, pp. 79-94.

[10]. Wüst K., Gervais A. Ethereum Eclipse Attacks. Report, ETH Zurich Research Collection, 2016, 7 p.
[11]. Chen T. et al. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks. In Proc. of the International Conference on Information Security Practice and Experience, 2017, pp. 3-24.
[12]. Luu L. et al. Making smart contracts smarter. In Proc. of the ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.
[13]. Dhillon V., Metcalf D., Hooper M. The DAO Hacked. In Blockchain Enabled Applications, Apress. Berkeley, CA, 2017, pp. 67-78.
[14]. Mayer H. ECDSA security in bitcoin and ethereum: a research survey. CoinFabrik, 2016. Available at https://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf, accessed 29.05.2018.
[15]. Marcus Y., Heilman E., Goldberg S. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. IACR Cryptology ePrint Archive, Available at https://eprint.iacr.org/2018/236.pdf, accessed 29.05.2018.
[16]. Dika A. Ethereum Smart Contracts: Security Vulnerabilities and Security Tools, Master's thesis, NTNU, 2017.
[17]. Wöhrer M., Zdun U. Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity. In Proc. of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, 8 p.
[18]. Biryukov A., Khovratovich D., Tikhomirov S. Findel: Secure Derivative Contracts for Ethereum. In Proc. of the International Conference on Financial Cryptography and Data Security, 2017, pp. 453-467.
[19]. Ross S. A. The economic theory of agency: The principal's problem. The American Economic Review, vol. 63, №. 2, 1973, pp. 134-139.
[20]. Eisenhardt K. M. Agency theory: An assessment and review. Academy of management review, vol. 14, № 1, 1989, pp, 57-74.
[21]. Gale D., Hellwig M. Incentive-compatible debt contracts: The one-period problem. The Review of Economic Studies, vol. 52, №. 4, 1985, pp. 647-663.
[22]. Bolton P., Dewatripont M. Contract theory. MIT press, 2005, 744 p.
[23]. Edelman B., Ostrovsky M., Schwarz M. Internet advertising and the generalized second-price auction: Selling billions of dollars' worth of keywords. American economic review, vol. 97, №. 1, 2007, pp. 242-259.
[24]. Roth A. E., Ockenfels A. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. American economic review, vol. 92, №. 4, 2002, pp. 1093-1103.
[25]. Greenstein S. How the internet became commercial: Innovation, privatization, and the birth of a new network. Princeton University Press, 2015, 488 p.
[26]. Moeen M., Agarwal R. Incubation of an industry: Heterogeneous knowledge bases and modes of value capture. Strategic Management Journal, vol. 38, №. 3, 2017, pp. 566-587.
[27]. Handy C. Trust and the virtual organization. Harvard business review, vol. 73, №. 3, 1995, pp. 40-51.
[28]. Markus M. L., Agres B. M. C. E. What makes a virtual organization work? MIT Sloan Management Review, vol. 42, №. 1. 2000, 16 p.
[29]. Szabo N. The idea of smart contracts. Nick Szabo's Papers and Concise Tutorials. Available at

http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_idea.html, accessed 29.05.2018.
[30]. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. Available at https://bitcoin.org/bitcoin.pdf, accessed 29.05.2018.
[31]. Haber S., Stornetta W. S. How to time-stamp a digital document. In Proc. of the Conference on the Theory and Application of Cryptography, 1990, pp. 437-455.
[32]. Massias H., Avila X. S., Quisquater J. J. Design of a secure timestamping service with minimal trust requirement. In Proc. of the 20th Symposium on Information Theory in the Benelux, 1999, pp. 79-86.
[33]. Merkle R. C. Protocols for public key cryptosystems. In Proc. of the IEEE Symposium on Security and Privacy, 1980, pp. 122-122.
[34]. Katz J. et al. Handbook of applied cryptography. CRC press, 1996, 810 p.
[35]. Özsu M. T., Valduriez P. Principles of distributed database systems. Springer Science & Business Media, 2011, 846 p.
[36]. Bernstein P. A., Hadzilacos V., Goodman N. Concurrency control and recovery in database systems. 1987. Available at https://www.microsoft.com/en-us/research/wp-content/uploads/2016/05/ccontrol.zip, accessed 29.05.2018.

Методы защиты децентрализованных автономных организаций от системных отказов и атак

А.А. Андрюхин <Alexandr@kcdigital.ru>
ООО "КЕЙСИДИ", 129226, г Москва, проспект Мира, 131, офис 3

Аннотация. В статье обсуждаются технология блокчейнов, децентрализованные автономные организации, смарт-контракты и их устойчивость к атакам и сбоям. Из-за того, что такая форма организаций является экспериментальной, их участники часто сталкиваются с проблемами атак на организацию, последствиями неправильно написанных правил и мошенничества. Задача создания децентрализованных автономных организаций, которые устойчивы к отказам и атакам, и исследование причин этих проблем стало актуальным для проектировщиков и разработчиков программного обеспечения. В статье исследуются алгоритмы атак и предлагаются методы обеспечения устойчивости децентрализованных автономных организаций для атак на основе анализа подпроцессов пограничных событий и журналов с использованием методов Process Mining. Методы, которые необходимо разработать, должны оперативно выявлять и предотвращать несоответствия между предполагаемым и фактическим поведением смарт-контрактов, которые приводят к таким ошибкам в функционировании, как пустые транзакции, увеличенное время обработки блоков и т. д.

Ключевые слова: блокчейн; децентрализованные автономные организации; анализ процессов; смарт-контракт; security

DOI: 10.15514/ISPRAS-2018-30(3)-11

Для цитирования: Андрюхин А.А. Методы защиты децентрализованных автономных организаций от системных отказов и атак. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 149-164 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-11

Список литературы

[1]. Williams J. The Seconomics (Security-Economics) Vulnerabilities of Decentralized Autonomous Organizations. Lecture Notes in Computer Science, vol. 10476, 2017, pp. 171-179.
[2]. Atzei N., Bartoletti M., Cimoli T. A survey of attacks on Ethereum smart contracts (SoK). In Proc. of the International Conference on Principles of Security and Trust, 2017, pp. 164-186.
[3]. Mehar M. et al. Understanding a Revolutionary and Flawed Grand Experiment in Blockchain: The DAO Attack. Available at SSRN: https://ssrn.com/abstract=3014782, accessed 29.05.2018.
[4]. DuPont Q. Experiments in algorithmic governance: A history and ethnography of "The DAO," a failed decentralized autonomous organization. In Bitcoin and Beyond: Cryptocurrencies, Blockchains and Global Governance, Routledge, 2017, 212 p.
[5]. Nikolic I. et al. Finding The Greedy, Prodigal, and Suicidal Contracts at Scale. arXiv preprint arXiv:1802.06038, 2018.
[6]. Grossman S. et al. Online detection of effectively callback free objects with applications to smart contracts. Proceedings of the ACM on Programming Languages, vol. 2, issue POPL, article 48, 2017, 20 p.
[7]. Gurfinkel A. et al. The SeaHorn verification framework. In Proc. of the International Conference on Computer Aided Verification, 2015, pp. 343-361.
[8]. Bhargavan K. et al. Formal verification of smart contracts. In Proc. of the ACM Workshop on Programming Languages and Analysis for Security, 2016, pp. 91-96.
[9]. Delmolino K. et al. Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab. In Proc. of the International Conference on Financial Cryptography and Data Security, 2016, pp. 79-94.
[10]. Wüst K., Gervais A. Ethereum Eclipse Attacks. Report, ETH Zurich Research Collection, 2016, 7 p.
[11]. Chen T. et al. An Adaptive Gas Cost Mechanism for Ethereum to Defend Against Under-Priced DoS Attacks. In Proc. of the International Conference on Information Security Practice and Experience, 2017, pp. 3-24.
[12]. Luu L. et al. Making smart contracts smarter. In Proc. of the ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254-269.
[13]. Dhillon V., Metcalf D., Hooper M. The DAO Hacked. In Blockchain Enabled Applications, Apress. Berkeley, CA, 2017, pp. 67-78.
[14]. Mayer H. ECDSA security in bitcoin and ethereum: a research survey. CoinFabrik, 2016. Режим доступа: https://blog.coinfabrik.com/wp-content/uploads/2016/06/ECDSA-Security-in-Bitcoin-and-Ethereum-a-Research-Survey.pdf, дата обращения 29.05.2018.
[15]. Marcus Y., Heilman E., Goldberg S. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. IACR Cryptology ePrint Archive, Режим доступа https://eprint.iacr.org/2018/236.pdf, дата обращения 29.05.2018.
[16]. Dika A. Ethereum Smart Contracts: Security Vulnerabilities and Security Tools, Master's thesis, NTNU, 2017.
[17]. Wöhrer M., Zdun U. Smart Contracts: Security Patterns in the Ethereum Ecosystem and Solidity. In Proc. of the International Workshop on Blockchain Oriented Software Engineering (IWBOSE), 2018, 8 p.

[18]. Biryukov A., Khovratovich D., Tikhomirov S. Findel: Secure Derivative Contracts for Ethereum. In Proc. of the International Conference on Financial Cryptography and Data Security, 2017, pp. 453-467.
[19]. Ross S. A. The economic theory of agency: The principal's problem. The American Economic Review, vol. 63, №. 2, 1973, pp. 134-139.
[20]. Eisenhardt K. M. Agency theory: An assessment and review. Academy of management review, vol. 14, № 1, 1989, pp, 57-74.
[21]. Gale D., Hellwig M. Incentive-compatible debt contracts: The one-period problem. The Review of Economic Studies, vol. 52, №. 4, 1985, pp. 647-663.
[22]. Bolton P., Dewatripont M. Contract theory. MIT press, 2005, 744 p.
[23]. Edelman B., Ostrovsky M., Schwarz M. Internet advertising and the generalized second-price auction: Selling billions of dollars' worth of keywords. American economic review, vol. 97, №. 1, 2007, pp. 242-259.
[24]. Roth A. E., Ockenfels A. Last-minute bidding and the rules for ending second-price auctions: Evidence from eBay and Amazon auctions on the Internet. American economic review, vol. 92, №. 4, 2002, pp. 1093-1103.
[25]. Greenstein S. How the internet became commercial: Innovation, privatization, and the birth of a new network. Princeton University Press, 2015, 488 p.
[26]. Moeen M., Agarwal R. Incubation of an industry: Heterogeneous knowledge bases and modes of value capture. Strategic Management Journal, vol. 38, №. 3, 2017, pp. 566-587.
[27]. Handy C. Trust and the virtual organization. Harvard business review, vol. 73, №. 3, 1995, pp. 40-51.
[28]. Markus M. L., Agres B. M. C. E. What makes a virtual organization work? MIT Sloan Management Review, vol. 42, №. 1. 2000, 16 p.
[29]. Szabo N. The idea of smart contracts. Nick Szabo's Papers and Concise Tutorials. Режим доступа: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_idea.html, дата обращения 29.05.2018.
[30]. Nakamoto S. Bitcoin: A peer-to-peer electronic cash system. 2008. Режим доступа https://bitcoin.org/bitcoin.pdf, дата обращения 29.05.2018.
[31]. Haber S., Stornetta W. S. How to time-stamp a digital document. In Proc. of the Conference on the Theory and Application of Cryptography, 1990, pp. 437-455.
[32]. Massias H., Avila X. S., Quisquater J. J. Design of a secure timestamping service with minimal trust requirement. In Proc. of the 20th Symposium on Information Theory in the Benelux, 1999, pp. 79-86.
[33]. Merkle R. C. Protocols for public key cryptosystems. In Proc. of the IEEE Symposium on Security and Privacy, 1980, pp. 122-122.
[34]. Katz J. et al. Handbook of applied cryptography. CRC press, 1996, 810 p.
[35]. Özsu M. T., Valduriez P. Principles of distributed database systems. Springer Science & Business Media, 2011, 846 p.
[36]. Bernstein P. A., Hadzilacos V., Goodman N. Concurrency control and recovery in database systems. 1987. Режим доступа https://www.microsoft.com/en-us/research/wp-content/uploads/2016/05/ccontrol.zip, дата обращения 29.05.2018.

Cryptographic Stack Machine Notation One

S.E. Prokopev <s.e.pr@mail.ru>
Independent researcher
Moscow, Russia

Abstract. A worthy cryptographic protocol specification has to be human-readable (declarative and concise), executable and formally verified in a sound model. Keeping in mind these requirements, we present a protocol message definition notation named CMN.1, which is based on an abstraction named *cryptographic stack machine.* The paper presents the syntax and semantics of CMN.1 and the principles of implementation of the CMN.1-based executable protocol specification language. The core language library (the engine) performs all the message processing, whereas a specification should only provide the declarative definitions of the messages. If an outcoming message must be formed, the engine takes the CMN.1 definition as input and produces the binary data in consistency with it. When an incoming message is received, the engine verifies the binary data with respect to the given CMN.1 definition memorizing all the information needed in the further actions. The verification is complete: the engine decrypts the ciphertexts, checks the message authentication codes and signatures, etc. Currently, the author's proof-of-concept implementation of the language (embedded in Haskell) can translate a CMN.1-based specifications both to the interoperable implementations and to the programs for the ProVerif protocol analyzer. The excerpts from the CMN.1-based TLS protocol specification and corresponding automatically generated ProVerif program are provided as an illustration.

Keywords: cryptographic stack machine; cryptographic protocol message notation; executable cryptographic protocol specification languages; embedded domain-specific languages; Haskell; ProVerif; TLS.

DOI: 10.15514/ISPRAS-2018-30(3)-12

For citation: Prokopev S.E. Cryptographic Stack Machine Notation One. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 165-182. DOI: 10.15514/ISPRAS-2018-30(3)-12

1. Introduction

The establishment of good soundness relations between cryptographic protocol implementations and their formal models is a popular research area. The existing approaches differ by the starting point of development (implementation first [1-6] or formal model first [7-9]), by the degree of cryptographic soundness of the models (symbolic [10] or computational [9]), by the presence of the formal proof of the soundness of the model-to-implementation (or vice verse) translation procedure, by implementation usability area and by other aspects.

Our aim is to soundly tie not two (implementation and formal model) but three elements of the protocol development process: implementation, formal model and specification. By the latter, we mean a human-readable protocol description that is usually placed in RFC. The models' languages, which are based on logics or special versions of general-purpose programming languages, are not quite suitable for this task: they are either not convenient for capturing the low-level details or are firmly imperative.

Therefore, our goal is a declarative specification language that could be directly used in the RFCs to considerably enhance the degree of formalization of these documents. Yet, the specification must be automatically translatable both to the interoperable implementation and to the programs for the state-of-the-art protocol model analyzers such as ProVerif [10] and Tamarin [11].

2. Related work

There exist many formal notations for data structures: ASN.1, JSON, etc. These notations are often provided with the engines, which can automatically generate the binary data using the provided data structure definition and, in the opposite direction, automatically unpack the binary data in accordance with the definition. Such projects as CSN.1 [12], TSN.1 [13], BinPAC [14], NetPDL [15] are targeted specifically at the network protocols.

While the readability of some of these notations can be suitable, their expressiveness (in the domain of cryptographic protocols) does not. We need to have behind the notation not simply a message generator/parser waiting to be embedded to some bigger program, but a generic cryptographic protocol implementation waiting for (semi-)declarative specification to adjust to specific case. Therefore, the primary challenge is to find such powerful underlying abstraction, whereas the notation would have to be naturally emerged from it.

3 Cryptographic Stack Machine Notation One

We propose an abstraction named *cryptographic stack machine* (abbreviated as CSM), which is a stack machine specifically tailored to the needs of cryptographic protocols. Within the proposed approach, the message definition is in fact a sequence of the CSM instructions. The instructions set is divided into "bare-metal" and "sugared" parts. The "sugared" instructions make the message definitions (which in their essence are imperative) looking declarative. The instructions set may be expanded if needed.

To reflect the fact that the declarative style of the protocol message definitions is one of the main targets, we name our notation «Cryptographic Stack Machine Notation One» (abbreviated as CMN.1) adopting the naming style of the ASN.1, CSN.1 and TSN.1 notations.

3.1 CMN.1 syntax

Below, the terms 'String', 'Integer', 'Int', 'Word8' denote the sets of strings, unlimited integers, integers ranged from 0 to 2^{32}-1 and integers ranged from 0 to 2^{8}-1, respectively. The curled brackets mean repetition, the square ones – optionality. The symbol ',' means comma itself, not concatenation.

Prog ::= "["{Instr,}*[Instr]"]"
Instr ::= BareMetal | Sugared
BareMetal ::= Const Word8List | Var VarName Role VarType | V VarName | SEnc' SEncAlg | Enco' EncoAlg | Xor' Int | ModAdd' | ModMult' | ModInv' | Add' Integer | Rev RFun | Hash' HashAlg | Pad' Int Word8List | Mod' | ModExp' | Take' IntList | Split' IntList | SplitE' Int | ECMult' | ECAdd' | C' | CE' | Len' LenHdr | InsertTo Int | PickFrom Int | Dup Int | Free Int | Elem Int Prog | Map' Prog Int Int | Sort' Int Int | SA' Int Int Prog | Select' CaseList | M Prog | L Int Inst
Sugared ::= C Prog | CE Prog | Hash HashAlg Prog | SEnc SEncAlg Prog | Enco EncoAlg Prog | Mod Prog | ModAdd Prog | ModMult Prog | ModExp Prog | ModInv Prog | ECMult Prog | ECAdd Prog | Len LenHdr Prog | Xor Prog | Add Integer Prog | Take IntList Prog | Split IntList Prog | SplitE Int Prog | Pad Int Word8List Prog | Map Prog Int Prog | Sort Int Prog | Select Inst CaseList | SA Prog | WithLen LenHdr Prog | VarL Int VarName Role VarType | VL Int VarName | SelectV VarName CaseList
VarName ::= "["{String,}*String"]"
VarType ::= Plain Int | Primary Int | Modulo Inst | UTC | ECx Inst | Sublist Prog | Choice Prog | Subset Prog | Is Prog
Word8List ::= "["{Word8,}*[Word8]"]"
IntList ::= "["{Int,}*Int"]"
IntegerList ::= "["{Integer,}*Integer"]"
SEncAlg ::= AES128CBC | AES256CBC …
HashAlg ::= SHA1 | SHA256 …
EncoAlg ::= SSLPad Int | B2DERInt | B2DERBits …
LenHdr ::= BE Int | LE Int | DER
CaseTy ::= Case Word8List Prog | Cases "["{Word8List,}*Word8List "]" Prog | Case' Condition Prog | Otherwise Prog | CaseUndef Prog
CaseList ::= "["{CaseTy,}*CaseTy"]"
Condition ::= Bytes Word8List | Equal Integer | Less Integer | More Integer | LessOrEq Integer | MoreOrEq Integer | OneOf IntegerList |Otherwise'
Role ::= Clnt | Serv | A | B | S | CA | RA | TTP …

3.2 CMN.1 semantics

CSM has one main stack and varying number of temporary stacks, random-number generator, real-time clock, the storage s_var containing the values of the protocol variables (actually they don't vary in CSM) and the register s_rol containing the identifier of the protocol role (fig. 1).

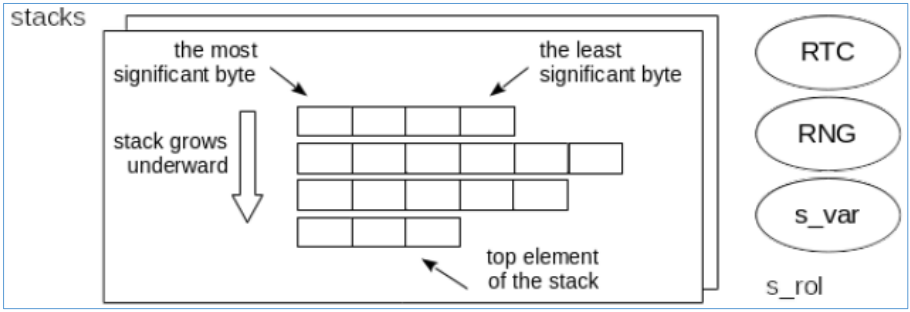


Fig. 1. Cryptographic stack machine

The language of the CSM instructions extends the line of the stack-oriented languages. It supports branching but doesn't support looping or recursing (table 1).

Table 1. CSM instructions semantics

Instruction	CSM actions
"Bare-metal" instructions	
Const *bs*	CSM pushes the byte string *bs* onto the stack.
Var *s r t*	If the storage s_var contains the variable named *s*, then CSM pushes this variable value onto the stack. Otherwise, if *r* /= s_rol, CSM returns an error. Otherwise, it generates a new element of type *t*, stores its value under the name *s* in the s_var storage and puts this value in the stack. The currently defined variable types: Plain *n* – random *n* bytes; Primary *n* – random primary integer of *n*-bit length; Modulo *is* – random integer modulo *n*, where *n* is the big-endian value of the result of the instruction *is* execution; ECx *is* – random point on the curve *curve_id*, where *curve_id* is the value of the result of the instruction *is* execution; UTC – the time and date in standard UNIX 32-bit format; Sublist *p* (Choice *p*, Subset *p*) – random sublist (element, subset) of the list comprised of resulting elements of the program *p* execution; Is *p* – equivalent to Choice [C *p*].
V *s*	If the storage s_var contains the variable with name *s*, then CSM pushes the value of this variable onto the stack. Otherwise, it returns an error.
SEnc' *alg*	CSM takes the top 3 elements of the stack as arguments: *a, b, c*. CSM encrypts *a* with *b* as initial vector and *c* as the key using symmetric encryption algorithm *alg*. Here and after: 1) if the stack is underflowed, CSM returns an error; 2) the last argument in the argument list is located at the top of the stack; 3) the arguments of the function are removed from the stack; 4) the result is pushed to the stack.
Enco' *alg*	Encoding of *a* using algorithm *alg*. List of arguments: *a*.

Xor' *n*	Exclusive OR. Arguments: the top *n* elements of the stack.
ModAdd', ModMult'	Addition (multiplication) of *a* and *b* modulo *m*. List of arguments: *a, b, m*. Here and after: the byte strings are interpreted as integers basing on the `big endian` agreement.
ModInv'	Inverse of *a* under modulo *m*. List of arguments: *a*, *m*.
Add' *n*	Let *a* is the top element of the stack. CSM adds *n* to *a* modulo *2^(8*k)*, where *k* is the length of *a* in bytes.
Rev *fun*	The function that is reverse to the function *fun*, where *fun* must be one of: Enco' *alg*, SEnc' *alg*, Xor' *n*, ModMult', ModAdd', ModInv', Add' *n*.
Mod'	Modulo operation. List of arguments: *a*.
ModExp'	Modular exponentiation: *a^b* mod *m*. List of arguments: *a,b,m*.
Hash' *alg*	CSM calculates the hash of *a* using algorithm *alg*. List of arguments: *a*.
Pad' *n ws*	Padding of *a* using the bytes *ws* until the length of the result reaches *n* (*n* must be equal or greater than length of *a*). List of arguments: *a*.
Take' *ns*	Here *ns* is the list of numbers. If the length of the top element of the stack is less than the sum of the elements of *ns*, then CSM returns the specification error. Otherwise, CSM cuts the top element of the stack into *n* parts considering the numbers from the *ns* list as lengths of elements and pushes (from left to right) the resulting *n* elements onto the stack, where *n* is the length of the *ns* list. The remainder of the top element is dropped (if any).
Split' *ns*	The same as the instruction Take' *ns*, except that the length of the top element of the stack must be exactly equal to the sum of the numbers from the *ns* list.
SplitE' *n*	Is equivalent to the instruction Split' [*k,k...k*], where *k = len / n*, where *len* is the length of the top element of the stack (*len* must be dividable by *n*).
ECMult'	Elliptic curve scalar multiplication. List of arguments: *curve_id* (curve identifier), *x* (x-coordinate), *y* (y-coordinate), *k* (the scalar). Instruction produces 2 elements of the stack: x-coordinate and y-coordinate.
ECAdd'	Elliptic curve addition of points (*x1,y1*) and (*x2,y2*). List of arguments: *curve_id* (curve identifier), *x1*, *y1*, *x2*, *y2*. Instruction produces 2 elements of the stack: x-coordinate and y-coordinate.
C' *n*	Concatenation. Arguments: the top *n* elements of the stack.
CE' *n*	Concatenation of the equal-sized arguments.
Len' *e*	The length of the top element of the stack written in *e* format, where *e* can be

	one of: BE *n* (packing into *n* big-endian bytes), LE n (packing into *n* little-endian bytes), DER (packing using ASN.1 DER format).
Insert *i*	CSM moves the top element of the stack to the *i*-th position.
Pick *i*, Dup *i*	CSM moves (for Pick) or copies (for Dup) the *i*-th element of the stack to the top position.
Free *i*	CSM removes the *i*-th element from the stack.
Elem i *p*	CSM executes the program *p* using temporary empty stack and then puts in the current working stack the *i*-th element of temporary stack.
SA' *n k p*	CSM copies *n* elements from the current working stack to temporary stack, executes the program *p* using a new temporary stack and then inserts the resulting elements between the *(k+1)*-th and *k*-th elements of the current working stack.
Map' *p i n*	The stack must contain at least *i*n* elements. CSM executes the program *p n* times using at each iteration a new temporary stack to which the next *i* elements from the current working stack are moved (beginning from the depths of the stack). At each iteration the elements containing in temporary stack after execution of *p* are moved to the current working stack.
Sort' *i n*	CSM considers the top *i*n* elements of the stack as a list of *n* elements, where each element, in turn, is a list of *i* elements. CSM sorts this list of *n* elements comparing their first (from the depths of the stack) elements.
Select' *cs*	CSM converts the list of the cases *cs* into the form: [Case' c_1 p_1, … , Case' c_n p_n]. If CSM finds in the list *cs* (from left to right) the condition c_i to which the top element of the stack satisfies, then it removes the top element from the stack and executes the program p_i. Otherwise, it returns an error.
M *p*	Macro instruction: CSM simply executes the program *p*.
L *n p*	Macro instruction supplemented by the total length of the resulting elements of *p* execution (parameter *n*).
"Sugared" instructions	
C *p,* CE *p,* Xor *p,* SEnc *al p,* Mod *p,* ModMult *p,* ModAdd *p,* ModExp *p,* ModInv *p,* ECMult *p,* ECAdd *p*	CSM executes the program *p* using temporary empty stack and copies the resulting *m* elements onto the current working stack. Then it executes the "bare-metal" counterpart of the "sugared" instruction: C' *m*, CE' *m*, Xor' *m*, SEnc' *al,* Mod', ModMult', ModAdd', ModExp', ModInv', ECMult' or ECAdd'. In the end, CSM moves the resulting elements (two elements in the case of the ECMult' or ECAdd' instruction and one element in the other cases) to the current working stack.

Map *q n p* Sort *n p*	CSM executes the program *p* using temporary empty stack. If *m* mod *n* /= *0*, CSM returns an error (where *m* is the number of elements of temporary stack after execution of *p*). Otherwise, it copies the resulting *m* elements onto the current working stack executes the "bare-metal" counterpart: Map' *q i n* or Sort' *i n*, where *i = m / n*.
Hash *al p*, Enco *al p*, Add *n p*, Pad *n bs p*, Len *e p*, Take *lst p*, Split *lst p*, SplitE *n p*	CSM executes the program [C *p*] using temporary empty stack and copies the resulting element onto the current working stack. After that, CSM executes the "bare-metal" counterpart of the "sugared" instruction: Hash' *al*, Enco' *al*, Add' *n*, Pad' *n bs*, Len' *e,* Take' *ls*, Split' *ls* or SplitE' *n.*
Select *is cs*	CSM tries to execute the program [C [*is*]] using temporary empty stack. If the program was successfully executed, CSM copies the resulting element onto the current working stack and executes the instruction Select' *cs*. If the execution failed (due to unknown variable), CSM checks if the list *cs* does contain the element CaseUnkno *p*. If so, CSM executes the program *p*, otherwise it returns an error.
VarL *n s r t*	Is equivalent to: L n (Var *s r t*)
VL *n s*	Is equivalent to: L n (V s)
SA *p*	Is equivalent to: SA' *1 0 p*
WithLen *e p*	Is equivalent to: M [C *p*, SA' *1 1* [Len' *e*]]
SelectV *s cs*	Is equivalent to: Select (V s) cs

4. Simple CMN.1-based ***specification*** language

The language presented below is simple in the sense that it doesn't capture the protocol automata in full. A specification consists of the CMN.1-based message definitions and a sequence of protocol actions with simple branching support (table 2).

Table 2. Protocol actions

Action	Description
roles *rlist*	The action sets the roles participating in the protocol. Each role runs its own CSM instance.
msg *src dst p*	The message with the CMN.1 definition *p* is transferred from the role *src* to the role *dst*.
set *r vvlist*	Here *vvlist* is the list of pairs of type (V *name*, *is*). For each pair, the action executes the CSM instruction *is* and includes the pair (*name, val*) in a storage

	s_var belonging to the CSM instance of the role *r*, where *val* is concatenation of the resulting elements of the execution of *is*.
select *r is acs*	This action provides a branching support in the same manner as the CSM instruction Select *is cs* does. The difference between the lists *cs* and *acs* is that *cs* consists of elements Case *value p*, where *p* is a CSM program, whereas *acs* consist of elements Case *value a*, where *a* is a sequence of protocol actions.
trusted *r id p*	This action takes from a trusted storage the binary data stored under the name *id* and processes these data using CMN.1 definition *p* and the CSM instance of the role *r*.
connect *r port addr*	If this action is present, the specification turns into the client implementation acting as the protocol role *role*. The action carries out the connection to a third-party server implementation listening on the port *port* of the IP-address *addr*.
accept *role port*	The specification turns into the server implementation acting as *role* and listening on the port *port*.
printPV printPV'	Both actions generate the ProVerif program corresponding to the protocol events that took place at the time of the call. The first action generates a full program, the second one ignores the lengths fields of messages and related events as non-essential in order to make this program more concise and productive.

Bearing in mind the elegant and concise syntax of the Haskell language and advantages of embedded domain-specific languages, we integrate our CMN.1-based specification language in Haskell.

As an illustration, we present an excerpt from the CMN.1-based specification of the TLS protocol (fig. 2; note that the order of declarations can be arbitrary in the Haskell language). A specification, which serves as source for this excerpt, comprises about 500 lines (the total for client and server) covering substantial part of the TLS v.1.2 protocol including four ciphersuites and X.509 certificates support and excluding extensions and renegotiations. The specification turned into the implementation (see the actions connect and accept in the table 2) was successfully tested for interoperability with the OpenSSL v.1.0.2o tool (both in the client and server roles).

```
tlsMsg m src =
   [VL 1 ["contentType",m],
    SelectV ["version","clntHello"]
       [CaseUnkno [VarL 2 ["version",m] src
                      (Choice [Const [0x03,0x03], Const [0x03,0x02],
                               Const [0x03,0x01]])],
        Otherwise [V ["version","clntHello"]]],
    WithLen (BE 2)
       [SelectV ["CCS",show src]
           [CaseUnkno [payload],
            Otherwise [payloadProtected]]]]
    where
    payload =
     SelectV ["contentType",m]
       [Case [0x14] [VarL 1 ["CCS",show src] src (Is [Const [0x01]])],
        Case [0x15] [VL 1 ["alertLevel",m],
                     VL 1 ["alertDescr",m]],
        Case [0x16] [Var ["hshkMsg",m] src (Is hshkMsg)],
        Case [0x17] [V ["dataContent",m]]]
        where
        hshkMsg =
         [VL 1 ["hshkType",m],
          WithLen (BE 3)
           [SelectV ["hshkType",m]
             [Case [0x01] clntHello,
              Case [0x02] servHello,
              Case [0x0b] servCert,
              Case [0x0c] servKeyExch,
              ...]]]
          where
          clntHello =
             [VarL 2 ["version","clntHello"] Clnt
                 (Choice [Const [0x03,0x03], Const [0x03,0x02],
                          Const [0x03,0x01]]),
              random Clnt,
              Const [0],
              WithLen (BE 2) [Var ["suites","clntHello"] Clnt
                 (Subset [Const [0x00,0x38], Const [0x00,0x32],
                          Const [0xc0,0x0a], Const [0xc0,0x09]])],
              WithLen (BE 1) [Const [0]],
              Var ["helloExt","clntHello"] Clnt (Choice [Const []])]

          servHello =
             [VarL 2 ["version","servHello"] Serv
                 (Is [V ["version","clntHello"]]),
              random Serv,
              WithLen (BE 1) [Var ["sessId","servHello"] Serv (Plain 32)],
```

```
47              Var ["suite","servHello"] Serv
48                 (Choice [SplitE 2 [V ["suites","clntHello"]]]),
49              VarL 1 ["compressAlg","servHello"] Serv
50                 (Choice [Const [0x00]])]
51           servCert = ...
52           servKeyExch =
53              [keyExchParams,
54              VarL 1 ["sigHashAlg","servKeyExch"] Serv
55                (Is [SelectV ["suite","servHello"]
56                       [Cases [[0x00,0x32],[0x00,0x38],
57                               [0xc0,0x09],[0xc0,0x0a]] [Const [0x02]]]]),
58              VarL 1 ["sigAlg","servKeyExch"] Serv
59                (Is [SelectV ["suite","servHello"]
60                       [Cases [[0x00,0x32],[0x00,0x38]] [Const [0x02]],
61                        Cases [[0xc0,0x09],[0xc0,0x0a]] [Const [0x03]]]]),
62              WithLen (BE 2)
63                 [mDER 0x30 [mDER 0x02 [sigPart 1],
64                             mDER 0x02 [sigPart 2]]]]
65              where
66              keyExchParams =
67                SelectV ["suite","servHello"]
68                     [Cases [[0x00,0x32],[0x00,0x38]] dh,
69                      Cases [[0xc0,0x09],[0xc0,0x0a]] ecdh]
70                where
71                dh = [WithLen (BE 2) [dhP],
72                      WithLen (BE 2) [dhG],
73                      WithLen (BE 2) [dhPubk Serv "servKeyExch"]]
74                ecdh = ...
75                sigPart i =
76                  SelectV ["sigAlg","servKeyExch"]
77                     [Case [0x02] [Elem i sig_dsa],
78                      Case [0x03] [Elem i sig_ecdsa]]
79                  where
80                  sig_dsa = mSigDSA [hash,p,q,g,x,k] where
81                     [p,q,g,x] = [V [x,"servCert"] | x <- ["dsaP","dsaQ",
82                                                           "dsaG","dsaX"]]
83                      k = Var ["dsaK","servCert"] Serv (Modulo p)
84                  sig_ecdsa = ...
85                  hash = ...
86             ...
87     random src =
88      C [Var ["time",show src] src UTC,
89         Var ["salt",show src] src (Plain 28)]
90
91     dhP = Var ["dhP","servKeyExch"] Serv (Primary 256)
92     dhG = Var ["dhG","servKeyExch"] Serv (Modulo dhP)
93     dhX src a = Var ["dhX",a] src (Modulo dhP)
94     dhPubk src a = ModExp [dhG, dhX src a, dhP]
95     ...
```

```
 96     payloadProtected = ...
 97
 98 mDER t p = C [Const [t], WithLen DER (f t)] where
 99     f 0x02 = [Enco B2DERInt p]
100     f 0x03 = [Enco B2DERBits p]
101     f _ = p
102 ...
103 mSigDSA [e,p,q,g,x,k] = [r,s] where
104     r = Mod [ModExp [g, k, p], q]
105     s = ModMult [ModAdd [ModMult [r, x, q], e, q],
106                  ModInv [k, q], q]
107 main =
108     roles [Clnt,Serv] >>=
109     -- connect Clnt 4433 0 >>= -- accept Serv 4433 >>= --
110     sendHandsh Clnt Serv [0x01] 1 >>=
111     sendHandsh Serv Clnt [0x02] 2 >>=
112     ...
113     sendHandsh Serv Clnt [0x0c] 4 >>=
114     ...
115     printPV'
116
117 sendHandsh src dst htype i ss =
118     set src [(V ["contentType",show i], Const [0x16]),
119              (V ["hshkType",show i], Const htype)] ss >>=
120     msg src dst [tlsMsg (show i) src]
```

Fig. 2. CMN.1-based specification of the TLS protocol (an excerpt)

5. Translation to the ProVerif program

The ProVerif program presented in the fig. 3 was generated automatically from the above specification (it is a console output of the call printPV'; see the line 115 in the fig. 2). This program corresponds to the protocol trace based on the ciphersuite TLS-DHE-DSS-WITH-AES-256-CBC-SHA. The program passed the ProVerif compiler checks without warnings. The events and queries of interest have to be inserted manually because CMN.1-based specifications do not contain such information.

```
 1 free c: channel.
 2 ...
 3 fun ModExp(bitstring,bitstring,bitstring): bitstring.
 4 const dhG_servKeyExch: bitstring [data].
 5 const dhP_servKeyExch: bitstring [data].
 6   equation forall x:bitstring,y:bitstring;
 7     ModExp(ModExp(dhG_servKeyExch,x,dhP_servKeyExch),y,dhP_servKeyExch) =
 8         ModExp(ModExp(dhG_servKeyExch,y,dhP_servKeyExch),x,dhP_servKeyExch).
 9 fun ModAdd(bitstring,bitstring,bitstring):bitstring.
10   equation forall a0:bitstring,a1:bitstring;
11     ModAdd(a0,a1,dhP_servKeyExch) = ModAdd(a1,a0,dhP_servKeyExch).
12   equation forall a0:bitstring,a1:bitstring;
13     ModAdd(a0,a1,dsaP servCert) = ModAdd(a1,a0,dsaP servCert).
```

```
14   reduc forall a0:bitstring,a1:bitstring,a2:bitstring;
15     Rev0ModAdd(ModAdd(a0,a1,a2),a1,a2) = a0.
16   reduc forall a0:bitstring,a1:bitstring,a2:bitstring;
17     Rev1ModAdd(a0,ModAdd(a0,a1,a2),a2) = a1.
18 fun ModInv(bitstring,bitstring):bitstring.
19   reduc forall a0:bitstring,a1:bitstring; Rev0ModInv(ModInv(a0,a1),a1) = a0.
20 fun HashSHA1(bitstring):bitstring.
21 fun Mod(bitstring,bitstring):bitstring.
22 fun EncoB2DERInt(bitstring):bitstring.
23   reduc forall a0:bitstring; Rev0EncoB2DERInt(EncoB2DERInt(a0)) = a0.
24 const xnull: bitstring [data].
25 const x0038: bitstring [data].
26 ...
27 let processClnt =
28   new time_Clnt: bitstring;
29   new salt_Clnt: bitstring;
30   let v17 = (time_Clnt,salt_Clnt) in
31   new suites_clntHello: bitstring;
32   let v25 = (x0303,v17,x00,suites_clntHello,x00,xnull) in
33   let hshkMsg_1 = (x01,v25) in
34   let v11 = (x16,x0303,hshkMsg_1) in
35   out(c,v11);
36   in(c,v37:bitstring);
37   let (=x16,=x0303,hshkMsg_2:bitstring) = v37 in
38   let (=x02,v48:bitstring) = hshkMsg_2 in
39   let (=x0303,v42:bitstring,sessId_servHello:bitstring,
40     =x0038,compressAlg_servHello:bitstring) = v48 in
41   let (time_Serv:bitstring,salt_Serv:bitstring) = v42 in
42   ...
43   in(c,v180:bitstring);
44   let (=x16,=x0303,hshkMsg_4:bitstring) = v180 in
45   let (=x0c,v217:bitstring) = hshkMsg_4 in
46   let (v193:bitstring,=x02,=x02,v214:bitstring) = v217 in
47   let (=dhP_servKeyExch,=dhG_servKeyExch,v190:bitstring) = v193 in
48   let (=x30,v211:bitstring) = v214 in
49   let (v206:bitstring,v210:bitstring) = v211 in
50   let (=x02,v203:bitstring) = v206 in
51   let v196 = Rev0EncoB2DERInt(v203) in
52   let (=x02,v207:bitstring) = v210 in
53   let v202 = Rev0EncoB2DERInt(v207) in
54   let v198 = (v17,v42,v193) in
55   let v199 = HashSHA1(v198) in
56   let v223 = ModInv(v202,dsaQ_servCert) in
57   let v224 = ModMult(v199,v223,dsaQ_servCert) in
58   let v226 = ModExp(dsaG_servCert,v224,dsaP_servCert) in
59   let v225 = ModMult(v196,v223,dsaQ_servCert) in
60   let v227 = ModExp(v132,v225,dsaP_servCert) in
61   let v230 = ModMult(v226,v227,dsaP_servCert) in
62   if v196 = Mod(v230,dsaQ_servCert) then
63   in(c,v237:bitstring);
64   ...
65 let processServ =
66   in(c,v10:bitstring);
```

```
 69  let (=x0303,v14:bitstring,=x00,suites_clntHello:bitstring,
 70    =x00,helloExt_clntHello:bitstring) = v22 in
 71  let (time_Clnt:bitstring,salt_Clnt:bitstring) = v14 in
 72  new time_Serv: bitstring;
 73  new salt_Serv: bitstring;
 74  let v38 = (time_Serv,salt_Serv) in
 75  new sessId_servHello: bitstring;
 76  if x0038 = Split2_2_2_2_1(suites_clntHello) then
 77  let v44 = (x0303,v38,sessId_servHello,x0038,x00) in
 78  let hshkMsg_2 = (x02,v44) in
 79  let v34 = (x16,x0303,hshkMsg_2) in
 80  out(c,v34);
 81  ...
 82  new dhX_servKeyExch: bitstring;
 83  let v203 = ModExp(dhG_servKeyExch,dhX_servKeyExch,dhP_servKeyExch) in
 84  let v206 = (dhP_servKeyExch,dhG_servKeyExch,v203) in
 85  new dsaK_servCert: bitstring;
 86  let v210 = ModExp(dsaG_servCert,dsaK_servCert,dsaP_servCert) in
 87  let v211 = Mod(v210,dsaQ_servCert) in
 88  let v218 = EncoB2DERInt(v211) in
 89  let v221 = (x02,v218) in
 90  let v212 = ModMult(v211,dsaX_servCert,dsaQ_servCert) in
 91  let v213 = (v14,v38,v206) in
 92  let v214 = HashSHA1(v213) in
 93  let v215 = ModAdd(v212,v214,dsaQ_servCert) in
 94  let v216 = ModInv(dsaK_servCert,dsaQ_servCert) in
 95  let v217 = ModMult(v215,v216,dsaQ_servCert) in
 96  let v222 = EncoB2DERInt(v217) in
 97  let v225 = (x02,v222) in
 98  let v226 = (v221,v225) in
 99  let v229 = (x30,v226) in
100  let v232 = (v206,x02,x02,v229) in
101  let hshkMsg_4 = (x0c,v232) in
102  let v194 = (x16,x0303,hshkMsg_4) in
103  out(c,v194);
104  ...
105 process
106  ((!processClnt) | (!processServ))
```

Fig. 3. The corresponding ProVerif program (an excerpt)

6. Engine implementation details

The engine implements the functionality that is significantly more powerful than the CSM machine presented in the section 3. The engine does not execute the CMN.1-notated programs as straightforward as CSM does. It executes the programs symbolically: the elements of the stack are not byte strings but symbolic expressions. This well-known technique allows the engine to fully take over the task of verification of the incoming messages using the same CMN.1-definitions that are used in the direct task of message generation. The verification is complete: the engine decrypts the ciphertexts, checks MACs and signatures, etc. Throughout a protocol execution, the engine accumulates the generated symbolic expressions, their values, lengths and types. It uses this information to generate or verify the

protocol messages in the future. In addition, the engine logs such events as calculations of the values of the symbolic expressions and applications of the rewriting rules. This information can be used by the engine's environment to extract symbolic traces and convert them to the programs for symbolic verifiers, e.g. ProVerif (as was presented in the previous section).

The scheme of the verification is as follows. Let the byte string *bs* is considered by the engine as a protocol message with the CMN.1 definition *p*. Let *EQ* is a set variable containing equations, i.e. pairs of type (symbolic expression, byte string). The engine implements the verification procedure as follows.

> **Step 1**. The engine executes the program *p* symbolically resulting the symbolic expression *exp*. *EQ* is initialized with the equation (*exp,bs*).
>
> **Step 2**. For every new equation (*exp,bs*) from *EQ*, until neither of Step 2.1 or Step 2.2 can be applied anymore:
>
> > **Step 2.1**. The engine tries to apply a rewriting rule to this equation. This rule can be a simple inversion (for Enco, SEnc, Xor, ModMult, ModAdd, ModInv or Add functions) or be a complex group operation taking into account other equations from *EQ* (e.g. for Split). The application of the rule produces one or several new equations, which are inserted in *EQ*. If some rule was applied, the engine returns to the beginning of the Step 2. Otherwise, it goes to the Step 2.2.
> >
> > **Step 2.2**. If the values of all the arguments of the top operation of the symbolic expression *exp* are known, the engine calculates the value of *exp*. If this value is equal to *bs*, the engine removes the equation from *EQ*. Otherwise, it returns the message verification error.

The engine knows about the equality $(a^b)^c = (a^c)^b$ and analogous equality for the elliptic curve scalar multiplication, so Diffie-Hellman key exchange and ElGamal asymmetric encryption do not ask for special treatment. Yet the engine uses specific rewrites for expressions relevant to the DSA and ECDSA algorithms or to their relatives.

The calls exported by the engine are presented below.

1. cSymExec *p* – The engine executes the program *p* symbolically and returns the descriptor of the generated symbolic expression.
2. cCalc *d* – The engine calculates the value of the symbolic expression with descriptor *d*.
3. cGetVal *d* – The engine returns the value of the symbolic expression with descriptor *d*.
4. cSetVal *d bs* – The engine assigns the value bs to the symbolic expression with descriptor d.
5. cVerify *d bs* – The engine verifies the byte string *bs* with respect to the symbolic expression with descriptor *d*. If verification has failed, it returns

an error, otherwise, it returns the superfluous remainder of the byte string *bs* (if present).

6. cEvent *ev* – The engine logs the event *ev* (i.e. the environment can insert additional events into the engine log).
7. cGetLog – The engine returns content of its log.

7. Conclusion

We presented cryptographic protocol message notation (named CMN.1) based on the instruction set of a stack machine specifically tailored to the needs of cryptographic protocols (named *cryptographic stack machine*, or CSM). The principles of implementation of the protocol specification language based on this notation also presented. Within such an approach, specifications are executable and also translatable to the programs for symbolic verifiers, such as ProVerif. The readability of CMN.1-notated specifications is brought in the court of public opinion.

In addition, the validation of the proposed notation on a wider spectrum of cryptographic protocols is needed. The validation will certainly cause minor additions to the notation (at least regarding cryptographic key types) without affecting currently defined CSM instructions. Taking into account the fact that the author's proof-of-concept implementation of the core language library (the engine) comprises only 700 lines of the Haskell code (excluding cryptographic primitives), it seems logical to provide in the future a formal description of the engine's algorithm and, basing on it, a proof of the soundness of the ProVerif-translation procedure.

References

[1]. S. Chaki and A. Datta. Aspier: An automated framework for verifying security protocol implementations. In Proceedings of the Computer Security Foundations Workshop, 2009, pp. 172–185.

[2]. J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05), Lecture Notes in Computer Science, vol. 3385, 2005, pp. 363–379.

[3]. Mihhail Aizatulin, Andrew D. Gordon, and Jan Jurjens. Extracting and verifying cryptographic models from C protocol code by symbolic execution. In Proc. of the 18th ACM Conference on Computer and Communications Security (CCS'11), 2011, pp. 331–340.

[4]. Nicholas O'Shea. Using Elyjah to analyse Java implementations of cryptographic protocols. In Proc. of the Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (FCS-ARSPA-WITS'08), 2008.

[5]. Karthikeyan Bhargavan, Cedric Fournet, Andrew Gordon, and Stephen Tse. Verified interoperable implementations of security protocols. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 31, no. 1, 2008.

[6]. Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In Proc. of the IEEE Symposium on Security and Privacy, 2017.
[7]. Matteo Avalle, Alfredo Pironti, Riccardo Sisto, and Davide Pozza. The JavaSPI framework for security protocol implementation. In Proc. of the 6th International Conference on Availability, Reliability and Security (ARES'11), 2011, pp. 746–751.
[8]. David Cade, Bruno Blanchet. From Computationally-Proved Protocol Specifications to Implementations and Application to SSH. Available at: http://prosecco.gforge.inria.fr/personal/dcade/CadeBlanchetJoWUA13.pdf, accessed 10.06.2018.
[9]. A. Delignat-Lavaud et al. Implementing and Proving the TLS 1.3 Record Layer. In Proc. of the 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 463-482.
[10]. Bruno Blanchet. Automatic Verification of Security Protocols in the Symbolic Model: the Verifier ProVerif. In Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures, Lecture Notes in Computer Science, vol. 8604, 2014, pp. 54-87.
[11]. Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. Source files and annotated RFC for TLS 1.3 analysis. (2017). Available at: https://tls13tamarin.github.io/TLS13Tamarin/, accessed 10.06.2018.
[12]. Concrete Syntax Notation One (CSN.1). Available at: http://csn1.info, accepted 10.06.2018.
[13]. Transfer Syntax Notation One (TSN.1). Available at: http://www.protomatics.com/tsn1.Html, accessed 10.06.2018..
[14]. The BinPAC language. Available at: https://www.bro.org/sphinx/components/binpac/README.html, accessed 10.06.2018..
[15]. Mario Baldi, Fulvio Risso. NetPDL: An Extensible XML-Based Language for Packet Header Description. Computer Networks, vol, 50, issue 5, 2006, pp. 688-706.

Нотация криптографической стековой машины версии один

С.Е. Прокопьев <s.e.pr@mail.ru>
г. Москва

Аннотация. Хорошая спецификация криптографического протокола должна легко восприниматься человеком (быть декларативной и лаконичной), быть исполнимой и пройти процедуру формальной верификации в некоторой адекватной модели. Нацеливаясь на эти требования, в статье предложена нотация CMN.1, предназначенная для описания сообщений криптографических протоколов и основанная на вычислительной абстракции под названием *криптографическая стековая машина* (CSM). Статья описывает синтаксис и семантику CMN.1, а также представляет результаты разработки языка спецификаций криптографических протоколов, построенного на основе данной нотации и встроенного в язык Haskell. В авторской реализации вся обработка сообщений инкапсулирована внутри базового библиотечного модуля, в то время как спецификация должна лишь дать декларативные определения этих сообщений. При формировании исходящего сообщения протокола базовый модуль берет описание данного сообщения в нотации CMN.1 и возвращает фрагмент данных, сгенерированный по этому описанию. При обработке входящего сообщения базовый модуль берет поступивший фрагмент данных и описание ожидаемого сообщения в

нотации CMN.1 и возвращает вердикт об их соответствии друг другу, извлекая и запоминая при этом все содержащиеся в сообщении данные, необходимые для формирования или верификации следующих сообщений протокола. Процесс верификации является полным: базовый модуль осуществляет расшифрование, проверку кодов аутентификации сообщений и значений цифровой подписи и т.д. Текущая реализация языка включает функции трансляции спецификаций в исполняемый код, совместимый с существующими программными реализациями протоколов, а также функции конвертации спецификаций в программы на входном языке анализатора протоколов ProVerif. В качестве иллюстрации приводятся выдержки из CMN.1-спецификации протокола TLS и соответствующей ей автоматически сгенерированной программы для ProVerif.

Ключевые слова: язык спецификаций криптографических протоколов; нотация сообщений криптографических протоколов; криптографическая стековая машина; встроенные предметно-ориентированные языки; Haskell; ProVerif; TLS

DOI: 10.15514/ISPRAS-2018-30(3)-12

Для цитирования: Прокопьев С.Е. Нотация криптографической стековой машины версии один. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 165-182 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-12

Список литературы

[1]. S. Chaki and A. Datta. Aspier: An automated framework for verifying security protocol implementations. In Proceedings of the Computer Security Foundations Workshop, 2009, pp. 172–185.

[2]. J. Goubault-Larrecq and F. Parrennes. Cryptographic protocol analysis on real C code. In Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'05), Lecture Notes in Computer Science, vol. 3385, 2005, pp. 363–379.

[3]. Mihhail Aizatulin, Andrew D. Gordon, and Jan Jurjens. Extracting and verifying cryptographic models from C protocol code by symbolic execution. In Proc. of the 18th ACM Conference on Computer and Communications Security (CCS'11), 2011, pp. 331–340.

[4]. Nicholas O'Shea. Using Elyjah to analyse Java implementations of cryptographic protocols. In Proc. of the Joint Workshop on Foundations of Computer Security, Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security (FCS-ARSPA-WITS'08), 2008.

[5]. Karthikeyan Bhargavan, Cedric Fournet, Andrew Gordon, and Stephen Tse. Verified interoperable implementations of security protocols. ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 31, no. 1, 2008.

[6]. Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate. In Proc. of the IEEE Symposium on Security and Privacy, 2017.

[7]. Matteo Avalle, Alfredo Pironti, Riccardo Sisto, and Davide Pozza. The JavaSPI framework for security protocol implementation. In Proc. of the 6th International Conference on Availability, Reliability and Security (ARES'11), 2011, pp. 746–751.

[8]. David Cade, Bruno Blanchet. From Computationally-Proved Protocol Specifications to Implementations and Application to SSH. Режим доступа: http://prosecco.gforge.inria.fr/personal/dcade/CadeBlanchetJoWUA13.pdf, дата обращения 10.06.2018.

[9]. A. Delignat-Lavaud et al. Implementing and Proving the TLS 1.3 Record Layer. In Proc. of the 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 463-482.

[10]. Bruno Blanchet. Automatic Verification of Security Protocols in the Symbolic Model: the Verifier ProVerif. In Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures, Lecture Notes in Computer Science, vol. 8604, 2014, pp. 54-87.

[11]. Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. 2017. Source files and annotated RFC for TLS 1.3 analysis. (2017). Режим доступа: https://tls13tamarin.github.io/TLS13Tamarin/, дата обращения 10.06.2018.

[12]. Concrete Syntax Notation One (CSN.1). Режим доступа: http://csn1.info, дата обращения 10.06.2018.

[13]. Transfer Syntax Notation One (TSN.1). Режим доступа: http://www.protomatics.com/tsn1. Html, дата обращения 10.06.2018.

[14]. The BinPAC language. Режим доступа: https://www.bro.org/sphinx/components/binpac/ README.html, дата обращения 10.06.2018..

[15]. Mario Baldi, Fulvio Risso. NetPDL: An Extensible XML-Based Language for Packet Header Description. Computer Networks, vol, 50, issue 5, 2006, pp. 688-706.

Construction of validation modules based on reference functional models in a standalone verification of communication subsystem

D.A. Lebedev <lebedev_d@mcst.ru>
I.A. Stotland <stotl_i@ mcst.ru >
MCST, 24 Vavilov st., Moscow, 119334, Russia

Abstract. The paper proposes some approaches to functional verification of microprocessor communication controllers based on developing layered UVM (Universal Verification Methodology) test systems. In modern microprocessor systems there are a lots of controllers operating with their own data types. Communication controllers support transferring and transformation data between microprocessor units. Such transformation must be carried out quickly and without data corruption for the correct functioning of the whole system. Communication controllers could carry additional functions such transmission values of copies of the system registers, address translation and others. Brief overview of verification tools and benefits of application standalone simulation based verification for checking the correctness of communication subsystems are marked out in the paper. We present the approaches of construction a standalone UVM-based verification environment with checking module implemented in external functional reference model. We also propose some techniques for checking the correctness of communication subsystems: checking multiple-clock controllers using parametrized clock generator, supporting of credit exchange mechanisms. Presented approaches were used to verify the communication subsystem — Host-Bridge — of Sparc V9 eight-core microprocessor developed by MCST. The difficulties discovered in the process of test system developing and its resolutions are described in the paper. The results of using presented solutions for verification of communicating subsystem controllers and further plan of the test system enhancement are considered.

Keywords: test system; communication controller; functional verification; Universal Verification Methodology (UVM); reference model.

DOI: 10.15514/ISPRAS-2018-30(3)-13

For citation: Lebedev D.A., Stotland I.A. Construction of validation modules based on reference functional models in a standalone verification of communication subsystem. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 183-194. DOI: 10.15514/ISPRAS-2018-30(3)-13

1. Introduction

The state of the art in microprocessors composition includes a variety of hardware controllers, which differ in complexity, speed rate, volume and types of data transmitted over them. The characteristics of data are continuously increasing. At the same time, verification costs are increasing, because the possibilities of verification methods are significantly lagging behind the development of microprocessor systems and, accordingly, the correctness checking requires more resources [1].

Each peripheral controller in the system could have its own data format. Converting data format is one of the functions of the interface communication controllers. The communication controllers can be a part of communication subsystem also known as northbridge. The northbridge typically handles communications among the CPU, I/O and in some cases RAM. Therefore, these transformations must be carried out quickly and without data loss. For this reason, the verification of communication controllers is an important step in the development of the microprocessor system.

The rest of the paper is organized as follows. Section 2 reviews the existing techniques for verifying communication controllers. Section 3 suggests an approach to the problem of developing test system. Section 4 describes a case study and the suggested approaches. Section 5 reveals results and Section 6 concludes the paper.

2. Functional verification of the communication controllers

It is necessary to simulate the operation of entire environment while providing standalone verification of a controller. This requires a test system, the development of which could be started at the earliest stages of whole microprocessor development, as soon as module specification and the RTL-model becomes available. Standalone verification allows detecting errors in the early stages of project. In addition, it helps to create complicated, critical and incorrect situations for the verified module. The achivement of such situations using system verification of the whole microprocessor model takes lot of resources. It is also important to note that the localization of the error is faster, what reduces the debugging time of the controller.

Due to its location between the CPU and the peripheral interface controller, the communication controller, in addition to its basic data format transformation function, could include copies of registers, buffers, FIFOs, parts of distributed control systems, and perform other additional functions. A number of these features should be taken into account in the standalone verification of communication controllers.

It is essential that, according to the classification proposed in [2], the properties of communication controllers include the absence of a pipeline, the absence of strict time (in the system clock frequency) restrictions on transaction processing and tagging of transmitted data. Accordingly, when the devices of this type are verified it is possible to use event-checking modules.

There are a number of methods to build a test system and implement a standalone verification of microprocessor controllers. Among them there is a tool created in the "MCST" named Alone-env, the development of the ISP RAS named C++TESKHW and methodology UVM [3]. The Alone-env tool simplifies implementation of standalone Verilog tests by creating test sequences in C++. Its library provides a wrapper-class over Verilog description of the verified module. Despite the relative simplicity of using Alone-env tool, there are some disadvantages: the lack of collecting coverage means, high requirements for the testing reference model and the inability to reuse the test system. One of the C++TESKHW tool features is availability of test generation based on the device state graph traversal. However, sometimes it is very hard to define all of the states of device and it needs high accuracy of documentation and checking reference model. UVM is the most widespread verification methodology developed by Accellera Systems [4]. UVM is a library with well-described tools for building portable and reusable testbenches and their components. The test system based on UVM can generate pseudo-random constrained input requests to cover all the possible states of the verified device. Most of well-known simulation tools (like Incisive, VCS, etc.) support the methodology. Moreover, most of VIP (Verification IP) support UVM-based interfaces. We also have a number of test systems and libraries already written and debugged. Therefore, we choose to use UVM for verification of RTL implemented modules of microprocessor systems.

Alone-env and C++TESKHW do not support UVM and we cannot use these tools for UVM-based test system development. UVM provides the universal approach for all types of devices to develop test systems. In this way, test systems are becoming more complex and worse in debugging. UVM also has no additional approaches for construction of validation modules based on reference functional models. Therefore, the main purpose of our investigation is to develop and extend the methods of standalone verification of communication controllers using UVM and program reference models.

3. Principles for testing communication controllers

Standalone verification of communication controllers can be carried out using simulation reference models that are part of the test systems - specially implemented software environment for the verified device. Test system functions includes:

- generating of input requests;
- monitoring of reactions from the verified device and the reference model;
- checking of reactions;
- forming a conclusion about the completeness of testing.

Uvm_sequence_item and uvm_sequence extension classes are defined to generate pseudorandom constrained impacts. The first one defines a set of variables that are required for serialization of set of impacts into a serial bit format. The second performs a single or multiply generating of a set of variables to transmit a request.

The request generated by the uvm_sequence_item object is processed by a special uvm_sequencer class and passed to the uvm_driver class. Uvm_driver produces a transformation of generated random requests into sequential bit-vectors in accordance with the interface exchange protocol. Uvm_monitor class is passive. It tracks changes in the interface of the verified device, indicating the appearance of input or output data, then packages the serial bit signals into the uvm_sequence_item format and transmits for further analysis to the checking blocks. To simplify the structure of the test environment perception, the uvm_driver, uvm_monitor, and uvm_sequencer are combined in the uvm_agent class, shown on fig.1.

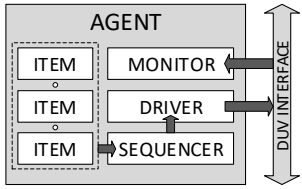


Fig 1. An example of the uvm_agent structure

Checking the reactions of the verified device can be carried out by internal means of the UVM library, however, if the verified device has a complex structure and many states, the checking module is based on the external to the controller environment reference model usually written in C++. A typical reference model-based test system is shown in Fig. 2.

In Fig. 2 DUV (Design Under Verification) is RTL-model of the verified device, ENV (Environment) - test environment. The number of agents are determined by the number of interface groups of the verified device (tracking the reactions uvm_monitor object can be taken outside from the agents). The reference model generates reference responses when impacts from the test environment are implied. Uvm_scoreboard is a checking module compares the response from the verification device and the reference model and makes a conclusion about the correctness of the operating. Using the DPI (Directed Programming Interface) of System Verilog is necessary to reconcile the types and classes of the test system written in SystemVerilog hardware description language with the C++ language. in which the reference model is developed.

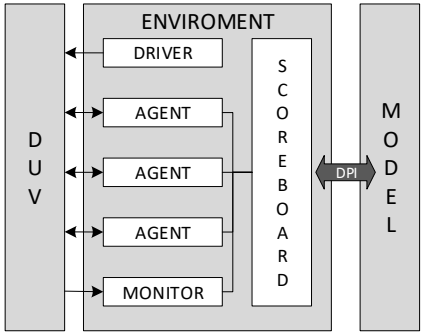


Fig 2. A typical structure of a test system for testing communications controllers.

The reference models could be divided into three types: cycle-accurate, discrete-event with time accounting and event models [5]. The choice of model type depends on the type of verified device, its architecture, and the complexity of developing a test environment. As stated earlier, the use of event models is justified for communication controllers, as they require less time to develop, maintain changes and can fully simulate the operation of such a controllers.

4. Functional verification of communication controller – Host-Bridge

Host-Bridge is a part of northbridge of microprocessor MCST-R2000 CPU with Sparc V9 architecture developing by MCST. The Host-bridge interface communication controller connects the system with external devices, accepting requests from the system and the I/O space while maintaining the transaction formats accepted in the system and I/O space. The Host-bridge receives requests from the System Commutator, communicate with two I/O channel controllers (IO-links) and provide translation of the virtual address to physical addresses. In addition, controller provides access to the system registers, registers of inter-processor links, memory controller's registers, transferring the new values of the registers to the local copies of them, transmitting interrupts, status signals and collecting snoop-responses. Each type of the registers has its own interface for communication with the destination device. All these features should be taken into account when verifying Host-bridge.

Some approaches for standalone verification using UVM and reference models were observed in [2, 6-8]. In [2] authors used buffers between testbench and reference model for checking marked transactions. In [6, 7] assertions and checking reference model were applied. In [8] there was reference model with very complicated algorithms. In our work, we used model buffers and assertions for checking correctness of transactions. In addition, we present a number of new solutions of the standalone verification process, which was applied for verification of Host-Bridge.

4.1 Several synchro signals parameters randomization

The main task of communication controllers is to coordinate the requests and data of several devices of the microprocessor system operating at different frequencies of the synchro signal. The parts of the communication controller in which several synchro signals interact should be checked carefully. Generating of random periods of synchro signals and their shifts that are relative to each other can be used for this purpose. The controller specification defines the operating ranges of each synchro signal. At the beginning of each test the frequency and start time of random synchro signal generators are pre-calculated. Thus, it is possible to detect errors in synchronization of internal units: the detection of metastability, desynchronization of releasing requests and data, sticking data in some positions of the buffers.

4.2 Support of credit exchange mechanisms

To control the flow of requests and free positions in the transaction buffers, Host-bridge supports a credit mechanism, which is a one-bit signal transmission that informs about the availability of free space in the buffers of connected devices. The management of this mechanism allows creating tests with full filling of all positions in the device buffers and needing to wait the vacated space to handle new requests, or on the other hand, tests, when the release of positions provides very fast, and the requests are executed almost instantly. As a result, it is possible to create test scenarios that are difficult to implement during system testing.

4.3 Verification of address translation controller

One of the components of the Host-Bridge is the address translation controller IOMMU. It translates a virtual address received from the I/O subsystem requests to physical addresses. The controller sends a request for information about the physical address to a special memory area for providing translation. Virtual address mapping to physical addresses is stored in a special controller buffer – IOTLB (Input-Output Translation Lookaside Buffer). If the buffer is full, the oldest element is displaced. The algorithm of the translation could be represented in the form of several consecutive steps:

1) receiving DMA request $p \leftarrow start(x)$,
2) analyzing of the input request then matching in the cache IOMMU with the following scenarios:
   a) match is found (hit IOMMU) – a request with the translated address x'' is executed;
   b) match is not found (miss IOMMU) – a request for a physical address x' is executed , then waiting for a response $p.receive(y)$ with the data, and only after that translation of the address is done.

Under dynamic test conditions, there may be situations when a lane in the IOMMU cache is not yet displaced in the RTL-model and the address can be translated without additional request, but it is not present in the reference model. In this case,

the reference model will give unnecessary requests to the test system. A global transaction counter was introduces in the reference model to solve this problem. The task of this counter was to identify the source of the requests. In addition, the responses generated by the test system are analyzed too. In the case when the request is successfully translated on the RTL-model side, and the reference model has already given an extra request for a physical address, the test environment generates a response that is marked with a special identifier and sent it to the reference model. When processing a response, the model concludes that the translation was not performed *p.model_check(y)*, calculates the desired transaction identifier *y.id* and sends it to a special buffer of canceled requests Q_{id} for the physical address. When checking the interchange buffers with reference model after the test completes, the identifier values in the buffer of canceled requests Q_{req} are compared with the identifiers of the remaining unprocessed requests for physical addresses from the reference model. Such remaining requests are not treated as erroneous and are deleted *delete(req.id)*. The pseudo-code of the algorithm is presented below.

DMA handling:

```
while true do
   wait p ← start(x)
   if x′ then
      p.ncheck(x′)
   else if x″ then
      begin
         wait p.receive(y)
         p.model_check(y)
         Q_id← y.id
         p.finish()
      end
end
```

After test checking:

```
for i ∈ Q_req do
   if c.check(req.id, Q_id) then
      delete(req.id)
   else report(req.id)
end
```

4.4 The correct organization of the exchange in terms of the uncertainty of the issuance of queries

In high-load dynamic tests with many input requests and responses, labeling requests and responses with tags that correspond to positions in the controller's buffers may differ from the values of tags in the reference model. This happens because of the inability to predict time of release of the buffer's position in the

event-driven models. Therefore, it is important to match the input and output requests of the model and the verified device. Each request, whether it is an I/O request or a PIO request, has several stages of execution. To ensure the correct functioning of the test scenarios we need to use an associative memory (mappers) for matching. The function of this memory is to store the matching of RTL request tags with reference model tags, when the remaining request data field, such as an address, destination device ID, processor number, and others are compared. Later, when you receive responses to the request, you have to pass to the model the same tag, which was allocated by the model at the stage of forming the request.

Communication controllers in multi-core systems can participate in the coherence protocol and accept snoop responses. Depending on the mode of operation and the content of the fields of the first received for the request response could come as several responses or only one. To complete the request check in coherent mode, it needed to pass all the responses with the correct tags to the model.

4.5 After test checking

The correct behavior of the communication controller is determined in providing a certain number of responses to requests and receiving the exact number of responses to them. Incorrect operation can be identified by counting the number of received requests, converted to another format requests, and accepted responses. For this purpose, the test system includes transaction counters. They capture all kinds of transactions while the test is running. At the end of the test, special algorithm checks values of these counters and make a conclusion about correctness.

After the test scenario is complete, it also needs to verify absence of unanswered requests in the buffers that link the reference model to the test environment. The presence of such requests signals about error of either the verified device or the reference model.

5. Results

The approaches described above were applied to standalone verification of the Host-Bridge of microprocessor MCST «R2000». Parametrization of synchro signals allowed finding metastability in the controller interfaces. Using different types of credit exchange rate helped to locate deadlocks and livelocks in the controller. Based on one test system the built-in IOMMU controller was also verified. Different configuration of answers with physical address were verified, which helps finding errors in displacement algorithms. After test, checking of model buffers and requests counters provide finding of not released responses to the system.

For the Host-Bridge controller, due to its functional and structural features that belong to the class of communication controllers, a test environment is developed with a checker based on reference event-model. Due to standalone verification of the device 67 errors that have not been found by other means of verification were found and corrected. Code and functional coverage was carried out and 94%

coverage was extracted. Gaps in coverage will be eliminated with the further expanding of the test environment with external parts of interrupt system. Total result indicates about effectiveness of standalone verification of communication controller.

6. Conclusion

Communication controllers are among the important parts of multi-core microprocessor systems have to be thoroughly tested. The principles described in the article do not depend mainly on the implementation of these controllers and allow their full standalone verification. The article suggests the ways to organize the interaction of the test system and the event reference model, as well as ways to resolve the difficulties encountered in the development of the test system.

The proposed approaches have been applied in the verification of the controller Host-bridge as a part of eight-core microprocessor, developed by "MCST". The developed test system and tests made it possible to detect and correct a number of logical errors that were not detected by other test methods.

In the future, it is planned to expand the test environment by adding a part of the interrupt system transmitting signals directly to the core. Fig. 3 shows a generalized diagram of a Host¬bridge, and the dotted lines illustrate such extension.

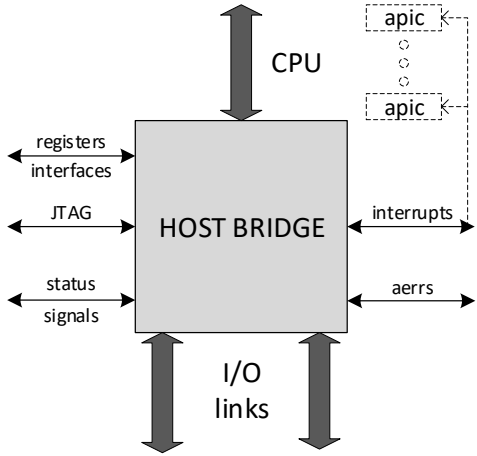


Fig 3. A typical structure of a Host Bridge controller connected with interrupt controllers

References

[1]. Kamkin A.M., Kotsynyak S.A, Smolov A.A.. Sortov A.D., Tatarnikov, Chupilko M.M. Tools for functional verification of microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 1, 2014, pp. 149-200 (in Russian).

[2]. Shmelev V.A., Stotland I.A. Standalone verification of microprocessors using reference models with various levels of abstraction. Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem [Problems of development of promising micro- and nanoelectronic systems], no 1, 2012, pp. 435-440 (in Russian)..
[3]. A.N. Meshkov, M.P. Ryzhov, V.A. Shmelev. The developement of the verification tools of the Elbrus-2S microprocessor. Voprosy radioelektroniki [Issues of radio electronics], ser. EVT, 2014, no. 3, pp. 5-17 (in Russian).
[4]. Standard Universal Verification Methodology http://accellera.org/downloads/standards/uvm (12.06.2018).
[5]. Kelton W., Law A. Simulation modeling. Classics of CS. 3rd ed. SPb.: Piter, 2004.
[6]. Li-Bo Cheng, Francis Anghinolfi, Ke Wang, Hong-Bo Zhu, Wei-Guo Lu, Zhen-An Liu. A UVM Based Testbench Research for ABCStar. In Proc. of the IEEE-NPSS Real Time Conference (RT), 2016. https://indico.cern.ch/event/390748/contributions/1825090/attachments/1280814/1906413/CR_PosterSession2_268.pdf (12.06.2018).
[7]. Abhineet Bhojak, Tejbal Prasad. A UVM Based Methodology for Processor Verification. Proc. of the Design and Verification Conference and Exhibition (DVCON), 2015. https://dvcon-india.org/sites/dvcon-india.org/files/archive/2015/proceedings/6_UVM_Based_Processor_Verification_paper.pdf (12.06.2018).
[8]. Kamkin A., Petrochenkov M. A Model-Based Approach to Design Test Oracles for Memory Subsystems of Multicore Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 149-160. DOI: 10.15514/ISPRAS-2015-27(3)-11.

Построение модулей проверки на основе эталонных функциональных моделей при автономной верификации подсистемы связи

Дмитрий Лебедев <lebedev_d@mcst.ru>
Ирина Стотланд <stotl_i@ mcst.ru >
АО «МЦСТ», 119334, Москва, Россия, ул. Вавилова, д. 24

Аннотация. В статье предложены подходы к функциональной верификации контроллеров сопряжения интерфейсов в составе микропроцессоров на основе разработки многоуровневых тестовых-систем по методологии UVM. В современных микропроцессорных системах существует множество контроллеров, работающих с собственными типами данных. Контроллеры сопряжения интерфейсов учувствуют в передаче и преобразовании данных между блоками микропроцессора. Такое преобразование должно осуществляться быстро и без повреждения данных для корректного функционирования всей системы. Контроллеры сопряжения интерфейсов могут выполнять дополнительные функции, такие как передача значений копий системных регистров, преобразование адресов и другие. В статье дан краткий обзор средств верификации и преимуществ применения автономной имитационной верификации для проверки корректности контроллеров сопряжения интерфейсов в составе подсистем связи. Представлены подходы к построению автономной

верификационной тестовой системы на основе методологии UVM с модулем проверки, реализованным во внешней функциональной эталонной модели. Так же предложены методы проверки корректности подсистем связи: проверка контроллеров, работающих с несколькими синхросигналами при помощи параметризованного генератора синхросигналов, поддержка механизмов обмена кредитами. Представленные подходы использованы для верификации подсистемы связи - Host-Bridge - восьмиядерного микропроцессора с архитектурой Sparc V9, разработанного компанией АО «МЦСТ». В статье описаны проблемы, обнаруженные в процессе разработки тестовой системы и способы их разрешения. Представлены результаты использования рассмотренных решений для верификации контроллеров подсистем связи и дальнейший план совершенствования тестовой системы.

Ключевые слова: тестовая система; контроллер сопряжения интерфейсов; функциональная верификация; Universal Verification Methodology (UVM); эталонная модель.

DOI: 10.15514/ISPRAS-2018-30(3)-13

Для цитирования: Лебедев Д.А., Стотланд И.А. Построение модулей проверки на основе эталонных функциональных моделей при автономной верификации подсистемы связи. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 183-194 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-13

Список литературы

[1]. А.С. Камкин, А.М. Коцыняк, С.А. Смолов, А.А. Сортов, А.Д. Татарников, М.М. Чупилко. Средства функциональной верификации микропроцессоров. Труды ИСП РАН, том 26, вып. 1, 2014, стр. 149-199.

[2]. Шмелев В.А., Стотланд И.А. Автономная верификация микропроцессоров на основе эталонных моделей разного уровня абстракции. Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС), No. 1, 2012, стр. 435-440.

[3]. А.Н. Мешков, М.П. Рыжов, В.А. Шмелев. Развитие средств верификации микропроцессора «Эльбрус-2S». Вопросы радиоэлектроники, сер. ЭВТ, вып.3, 2014, стр. С. 5-17.

[4]. Standard Universal Verification Methodology http://accellera.org/downloads/standards/uvm (12.06.2018).

[5]. Кельтон В., Лоу А. Имитационное моделирование. Классика CS. 3-е изд. СПб.: Питер, 2004.

[6]. Li-Bo Cheng, Francis Anghinolfi, Ke Wang, Hong-Bo Zhu, Wei-Guo Lu, Zhen-An Liu. A UVM Based Testbench Research for ABCStar. In Proc. of the IEEE-NPSS Real Time Conference (RT), 2016. https://indico.cern.ch/event/390748/contributions/1825090/attachments/1280814/1906413/CR_PosterSession2_268.pdf (12.06.2018).

[7]. Abhineet Bhojak, Tejbal Prasad. A UVM Based Methodology for Processor Verification. Proc. of the Design and Verification Conference and Exhibition (DVCON), 2015. https://dvcon-india.org/sites/dvcon-

india.org/files/archive/2015/proceedings/6_UVM_Based_Processor_Verification_paper.pdf (12.06.2018).

[8]. Kamkin A., Petrochenkov M. A Model-Based Approach to Design Test Oracles for Memory Subsystems of Multicore Microprocessors. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 149-160. DOI: 10.15514/ISPRAS-2015-27(3)-11.

Verification of System on Chip Integrated Communication Controllers

M.V. Petrochenkov <petroch_m@mcst.ru>
R.E. Mushtakov <mushtakov_r@mcst.ru>
D.I. Shpagilev <shpagilev_d@mcst.ru>
MCST, 1 Nagatinskaya st., Moscow, 117105, Russia

Abstract. This article presents an approach used to verify communication controllers developed for Systems on Chip (SOC) at MCST. We provide a list of communication controllers developed in MCST and present their characteristics. We describe principles of communication controller's operation on transaction, data link and physical layers and highlight their similarities. Then we describe a common method of device verification: principles of test system design, constrained random test stimuli generation and checking of device behavior. Based on common features of the controllers, we provide the general design of their test system. It includes components to work with transaction level interface (system agent of system on chip communication protocol) and physical interface (physical agent of protocol for SOC communication on a single board), configuration agent that determines device mode of operation and a scoreboard. Because controllers only execute transformation of transactions between different representation, scoreboard checks accordance of in and outgoing transactions. In addition, we describe specific features of devices that require the adjustments to the common approach. We describe how verification of those features affected the design of different test systems. We explain how a replacement of a physical agent with a second communication controller allows to speed up the development of test systems. We explain challenges of link training and status state machine (LTSSM) verification. We provide a way to work with devices with direct memory access (DMA) in a system agent. In conclusion, we present a list of found errors and directions of further research.

Keywords: Elbrus; system on chip; communication controller; Ethernet; DDR4; PCI Express; UVM; stand-alone verification

DOI: 10.15514/ISPRAS-2018-30(3)-14

For citation: Petrochenkov M.V., Mushtakov R.E., Shpagilev D.I. Verification of System on Chip Integrated Communication Controllers. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 195-206. DOI: 10.15514/ISPRAS-2018-30(3)-14

1. Introduction

Modern systems on chip (SOC) may include multiple microprocessor cores, complex hierarchy of caches, peripheral controllers and other types of data processing modules. The task of interconnection between different systems on chip is solved by communication controller (CC) modules. Those modules solve the

problem of interprocessor communications, communication between CPU and random access memory (RAM), CPU and peripheral devices, network interfaces, etc. Performance and reliability of communication controllers is crucial for the quality of the whole system. To ensure that communication controllers satisfy all requirements, they must be thoroughly verified. Verification of complex communication controllers is a time-consuming task [1]. One of the widely used approaches to verification of SOC is system verification - execution of test programs (implemented in assembly language) on the model of microprocessor. Another approach is stand-alone verification of SOC components. In this approach, model of the device under verification (DUT) is included in a special program – a test system, which goal is to ensure that DUT satisfies all requirements. This article describes a problem of stand-alone verification of communication controllers with physical media access interfaces in the industrial setting.

The rest of the paper is organized as follows. Section 2 describes communication controllers for physical media access interfaces developed by MCST company. Section 3 presents a common approach to the design a test system and describes its components. In section 4 we provide a case study for suggested approach applied to specific devices, and adjustments to the approach that were implemented to verify specific features of those devices. In conclusion, we present of verification and provide a direction of further research.

2. Overview of communication controllers in «Elbrus-16C» microprocessor

“Elbrus-16C” System on Chip includes many communication controllers. In the following list we will describe ones that require the stand-alone verification: the most complex ones and the ones which reliability is crucial for the functionality of the system.

1. DDR4 Memory Controller is a digital circuit that manages the flow of data going to and from the computer's main memory. The controller contains the logical circuits necessary to perform read and write operations in DRAM, with all necessary delays (for example, between reading and writing). The flow of incoming requests is converted into sequences of DRAM commands, while monitoring various conflicts on banks, buses and channels. To increase the effective bandwidth of the memory channel, incoming requests can be buffered and reordered. The reordering mechanism is implemented on the basis of a sequential combination filter system.
2. PCI Express Root Complex (RC) Controller transforms packets from in-house protocol to standard PCI Express transaction level packets and implements RC configuration space for communication with peripheral devices. The controller is connected directly to on-chip network to improve throughput and reduce delays. The controller supports up to 16 lanes with speed up to 8 GT/s [2].

3. Inter-Processor Communication Controller (IPCC) is designed to solve problems of organization of multiprocessor architectures with shared memory [3]. IPCC functions are logically divided into two levels: the link layer (DLL - Data Link Layer) and the physical layer (PHL - Physical Layer). Exchange by link is carried out by transport packages (containers) of fixed size. Packages contain information about the type of the channel, data, as well as the CRC checksum. Packages are formed into containers according to special rules in order to ensure the priority and maximize the bandwidth of the link. The protocol packets are distributed among several virtual channels (VC) or streams with different priorities. To ensure the integrity of the data during the transmission over the link, the mechanism of sequential container numbering and CRC encoding are used.
4. Wide Link Communication Controller (WLCC) is used to connect south bridge controller to SOC using a protocol similar to PCI Express 2.0 but with reduced overhead. Controller supports memory and configuration space access operations. Supported link width is up to 16 lanes with speed 2.5 or 5 GT/s for each lane. To ensure channel reliability transmitted packets are protected by 16 bit CRC. After transmission, packets are stored in replay buffer waiting for receive confirmation. If negative packet acknowledge is received or time-out is reached, packets are retransmitted. Controller supports up to 8 virtual channels.
5. 10 Gigabit Ethernet Controller uses 10GBASE-KR interface [4]. It sends and receives Ethernet frames over backplane electrical interface. On a physical layer, it supports procedures of Clause 73 Auto-negotiation and Clause 72 Auto-adaptation. This device supports hardware calculation and checking of Ethernet CRC, IPv4, TCP and UDP checksums, various filtering mechanisms based on MAC addresses and VLAN tags and automatic handling of pause frames.
6. Gigabit Ethernet Controller uses 1000BASE-KX interface [4]. Ethernet frames are sent using backplane electrical interface. It supports calculation and checking of Ethernet frame CRC, calculation and checking of IPv4, TCP and UDP checksums, filtering based on mac and IP addresses and automatic handling of pause frames.

Despite the fact that those devices implement sufficiently different protocols, they nonetheless solve a lot of similar problems and implement similar features. Common features of controllers are:

- Register transfer level (RTL) models of this devices are implemented using Verilog and SystemVerilog [5] hardware description languages.
- Controllers communicate with other components on chip using the system interface that implements on-chip communication protocol, and represents transaction layer of the device.

- Controllers don't possess complex internal state and don't implement complex data processing or caching mechanisms. They transform packets between different representations: system level communication protocol packets (used for on-chip communications) and physical interface signals (used for communication on distances beyond the single chip).
- Controllers implement data link layer (DLL) that performs error detection and/or correction using such mechanisms as Cyclic Redundancy Checks (CRC) or forward error correction (FEC).
- Controllers expose the physical interface and implement logical and electrical parts of physical layer for communication with other components on a board. All aforementioned controllers communicate using low voltage differential signaling (LVDS). To ensure clock recovery and dc balancing devices use physical encoding schemes (for example 8b/10b, 64b/66b, 128b/130b) and signal scrambling.

3. Test system structure

Test systems are usually implemented using either general purpose programming languages (C++), hardware description languages (VHDL, Verilog) or dedicated verification languages (SystemVerilog, e, OpenVera). In our company we use SystemVerilog [5] with Universal Verification Methodology [6] (UVM). Use of this language allows for an easy interface with Verilog and SystemVerilog devices, and UVM describes a general test system structure and provides a library of basic verification components.

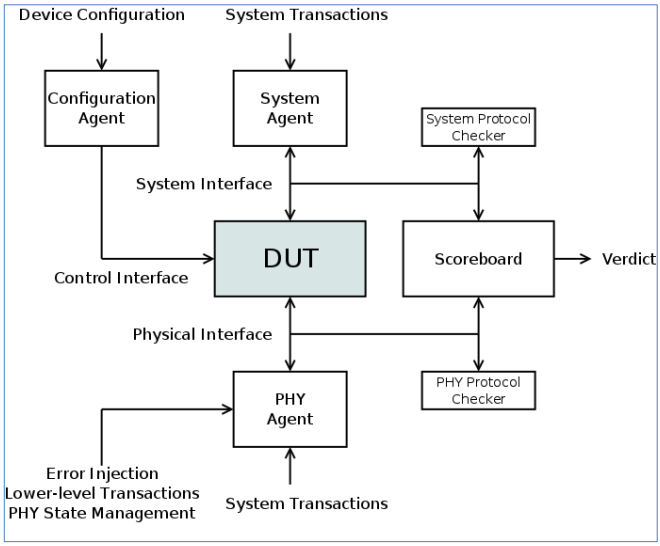


Fig 1. Structure of test system of communication controllers

Common principles of controller behaviour determine the general structure of the test system. All test systems include a set of basic components. Test system structure is presented in fig.1.

A. Test stimuli generators are based on constrained randomization. In our case, stimuli generators communicate with system and physical interfaces of DUT. Transactions are described in terms of their attributes and constraints. To specify some test scenario, one must define specific constraints for transactions that will be issued by request generators. SystemVerilog offers a native support for constrained randomization constructs. In addition to transaction transmission and reception. physical agent is able to model some “non-standard” types of behavior: injection of corrupted or non-standard compliant transactions, or handling of received transactions in user-specified way (for example, send negative acknowledge for non-corrupted packet, drop the response to request from DUT, etc..).
B. Test system scoreboard implements a correctness checks. Devices under verification do not possess complex data processing logic and simply perform transformation of transactions between different representations. Scoreboard receives transactions from system and physical interface monitors and performs comparison between ingress and egress transaction. If discrepancy between expected (transmitted) and received packets is detected, module reports an error in the test system.
C. In addition to global test system scoreboard, test system contains local (system and physical) interface protocol checkers. Their goal is to check that interface rules and invariants are not violated and otherwise report an error.
D. Configuration agent is used to access a set of memory-mapped configuration registers in the controllers. Those registers are accessed using separate configuration interface. Initial phase of a test is writing desired values to this registers.

4. Case study

This chapter describes the adjustment and highlights specific implementation details of different test systems.

4.1 Verification of Link Training and Status State Machine

One of the features of PCI Express, WLCC and IPCC links is a complex procedure of link initialization and training. During the initialization procedure device sends data patterns containing device capabilities and its current state across the link. Those data patterns are called a training sequence (TS). At the same time, using information from received training sequences, the controller detects the presence of a link partner, determines its active lanes and abilities. Based on this information,

pair of devices establishes common mode of operation for transaction transfer. In addition, training sequences are used to change the state of the link (for example, from active to low power mode or to the disabled state).

Presence of the LTSSM provides several additional challenges for the device verification.

- To send the transactions across the link, the active link must be established first. Thus, first action that the controller and its physical link agent partner performs is a link training sequence.
- One must test ability of the device to change its state and check that it reacts correctly to the state change of the link partner.
- In addition to "main" device states there are several "transient" states that the device passes when switching from one main state to another. Depending on training sequences received from link partner in transient states, link training procedure either continues successfully or terminates while reporting the error status.

It should be said that, despite the internal complexity of LTSSM protocols, they are almost invisible to the transaction layer. Only information available to transaction layer is whenever link is currently active or not.

4.2 Test systems based on a pair of controllers

To verify implementations of in-house communication protocols (IPCC and WLCC) additional type of test system was used [7]. It is based on the pair of RTL-models of communication controllers. In these test systems two controllers are connected using their corresponding physical interfaces. Errors are injected by manipulating the signals of physical interface. The structure of the test system is presented in fig.2.

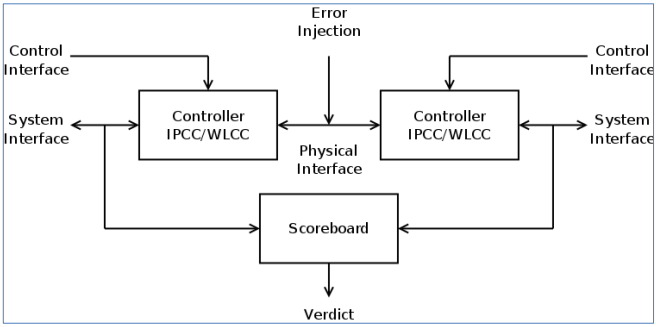


Fig 2. Structure of test system based on a pair of controllers

Advantages of the approach are as following.

- Simulation of device behaviour in realistic scenarios. Those devices (IPCC and WLCC) use our company's proprietary protocols to connect identical

devices, developed in-house. Thus, test system of this kind represents a realistic use-case of the device.

- Simplicity of implementation. The development of physical level agent is a labor-intensive and time-consuming, and its development cannot be avoided by purchasing a third party Verification IP (VIP). In this approach, the development of only a system agent is necessary, and verification can start earlier.

Disadvantages are as following.

- Lower simulation performance is caused by the need to simulate two identical controllers. This doubles the required computational resources.
- More difficult state and error injection control. To inject errors into sent and received transactions one must either directly manipulate external signals of the controller or use hierarchical access to modify the behaviour of the controllers.
- Inability to detect “self-correcting” bugs (for example, incorrect CRC polynomial). This disadvantage is mitigated by the fact those bugs will also self-correct in “real” device.
- Absence of checks on lower protocol levels. The main way to detect an error is to receive an unexpected packet on system interfaces. This may cause difficulties in bug detection and localization in many cases. For example, an error that causes an incorrect request to repeat a transaction can be detected only by performance degradation.

One can reduce the disadvantages while keeping most of some of the advantages of the approach by adding physical monitor on a link between devices.

4.3 Complex system agent in the Ethernet test systems

Distinctive feature of Ethernet test systems (both 10 Gigabit and Gigabit) is a complex system agent [8]. To reduce CPU usage and increase device efficiency controllers implement Direct Memory Access (DMA). Instead of sending Ethernet frames directly to device interfaces, frames are stored in system memory and the device reads the memory when it is ready for frame transmission. In a same way, the system must prepare a memory space for device to store received frames The device will write the data to this location after the frame reception. Ethernet controllers are managed using a set of memory-mapped registers. The most important ones are descriptor pointer registers (head and tail). Descriptors contain an Ethernet frame metadata (size of frame, memory location address, higher-level protocol information, etc...). The head register points to the first descriptor available to the controller, and the tail points to the last processed by it. Using those registers the controller reads and writes transaction descriptors and a frame memory. The structure of Ethernet agents is presented in fig. 3.

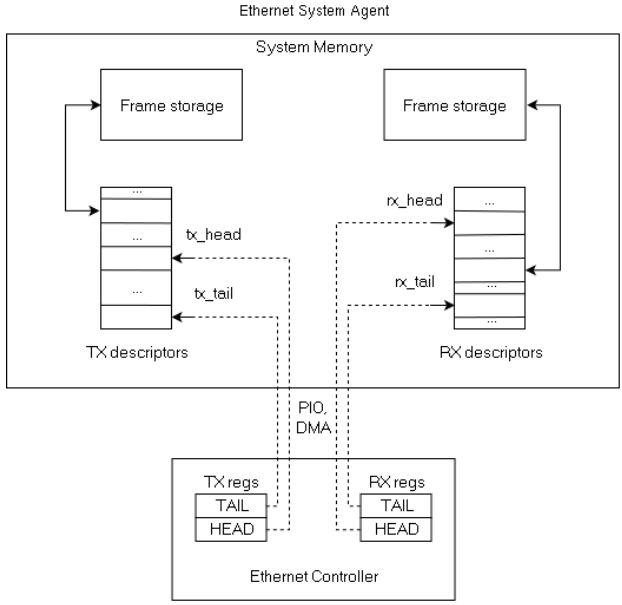


Fig 3. Structure of Ethernet controller test system

4.4 DDR4 Memory Controller protocol checks

A system agent in the memory controller test system consists of a set of two modules: the management agent of the information written into the memory and the agent for transferring requests from the system to the controller. The test system requires more sophisticated physical protocol checkers. For this purpose, two modules are used: the DFI protocol verification module and the DDR protocol verification module.

Before active work with the memory is started, the controller performs programming of the operating modes of the DRAM memory modules, conducts its initialization and training. To verify these processes, the DDR Protocol Checker is used. In addition to the fact that the module monitors the initialization and training of the memory, it also controls the execution of all the time constraints imposed to the controller when it issues commands to the memory.

Another important function of the memory controller is to periodically update the data stored in the DRAM using a refresh command. Without periodic updates, DRAM memory chips would gradually lose information, as capacitors storing bits are discharged by leakage currents. DDR protocol checker is used to analyze transactions on physical interface and to check if Refresh commands are issued within specified timing constraints. In addition, the memory state is checked before executing the Refresh command. The memory must be in the IDLE state. The controller has built-in noise immunity mechanisms that allow to check the integrity

of the data, and to correct it if necessary. Such mechanisms include: rectification of parity errors of the DDR bus, calculation of checksums, correction of CRC errors on the data bus of the DFI interface while writing, and correction of ECC errors on the DFI data bus during reading. Verification of noise immunity of transmitted data is provided by the DFI Protocol Checker. In addition, checker provides a way to verify the process of switching to and from power saving modes of memory chips by checking their timing parameters.

5. Conclusion

Methods described in the paper were used to verify components of "Elbrus-16C" microprocessor. Errors found in the controllers as a result of stand-alone verification are presented in table 1.

Table 1. Results of stand-alone verification

Device	Number of bugs
DDR4 MC	32
PCI Express RC	48
IPCC	13
WLCC	2
10 Gigabit Ethernet	51
Gigabit Ethernet	22

Verification of those devices is still ongoing. Our future work is aimed at improving those test systems, developing additional test scenarios and using the approach to verify other devices.

References

[1] Stotland I., Shpagilev D., Starikovskaya N. UVM based approaches to functional verification of communication controllers of microprocessor systems. In Proc. of the 2016 IEEE East-West Design & Test Symposium (EWDTS).

[2] PCI Express Base Specification Revision 3.0, http://pcisig.com/specifications

[3] Belyanin I, Petrakov P., Feldman V. Functional organization and hardware means of network interconnection of modules in computer cluster on «Elbrus» microproseccors. Voprosy radioelektroniki [Issues of radio electronics], ser. EVT, no. 3, 2015, pp. 7–20 (in Russian).

[4] IEEE Standard for Ethernet. IEEE Std 802.3-2012. 1983 p.

[5] IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800-2012

[6] 1800.2-2017 - IEEE Standard for Universal Verification Methodology Language Reference Manual

[7] Stotland I., Shpagilev D., Petrochenkov M. Features of High Speed Communication Controllers Standalone Verification of «Elbrus» Microprocessor Systems. Voprosy radioelektroniki [Issues of radio electronics], ser. EVT, 3, 2017, pp. 69-75 (in Russian).

[8] S. Chitti, P. Chandrasekhar, M. Asha Rani. Gigabit Ethernet Verification using Efficient Verification Methodology. In Proc. of the International Conference on Industrial Instruments and Control (ICIC), 2015, pp.1231-1235.

Верификация контроллеров связи в системах на кристалле

М.В. Петроченков <petroch_m@mcst.ru>
Р.Е. Муштаков <mushtakov_r@mcst.ru>
Д.И. Шпагилев <shpagilev_d@mcst.ru>
АО «МЦСТ», 117105, Россия, г. Москва, ул. Нагатинская, д.1, стр.1

Аннотация. В статье описаны подходы, которые использовались для верификации контроллеров связи в системах на кристалле, разрабатываемых в МЦСТ. Представлен список контроллеров связи, а также их характеристики. Приведены принципы работы контроллеров на уровне транзакций, канальном и физическом, и отмечен их общий функционал. Затем описан общий подход к верификации устройств: принцип проектирования тестовой системы, генерации случайных тестовых воздействий и проверки поведения устройства. Представлена общая структура тестовой системы, основанная на общих свойствах устройств. Она включает компоненты для работы с интерфейсом уровня транзакций (системный агент, реализующий коммуникационный протокол системы на кристалле), интерфейсом физического уровня (физический агент, реализующий коммуникационный протокол между различными системами на кристалле на одной плате), модуль конфигурационного интерфейса, определяющего режим работы устройства, а также модуль проверки. Отмечено, что поскольку устройства исполняют только преобразования транзакций между различными представлениями, заключение о корректности поведения осуществляется на основании простой проверки совпадения входящих и исходящих транзакций. Кроме того, приведены особенности функционала устройств, которые требуют адаптации общего подхода. Объяснено, как верификация данных особенностей работы устройств определила детали структуры тестовых систем. Описано, как замена физического агента на второй контроллер связи позволяет ускорить разработку тестовой системы. Представлены методы и сложности верификации конечного автомата тренировки и состояния линка (LTSSM). Описана структура и принцип работы системных агентов, поддерживающих прямой доступ к памяти (DMA). В заключение приведен список найденных ошибок и направления дальнейшей работы.

Ключевые слова: Эльбрус; система на кристалле; контроллер связи; Ethernet; DDR4; PCI Express; UVM; автономная верификация

DOI: 10.15514/ISPRAS-2018-30(3)-14

Для цитирования: Петроченков М.В., Муштаков Р.Е., Шпагилев Д.И. Верификация контроллеров связи в системах на кристалле. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 195-206 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-14

Список литературы

[1] Stotland I., Shpagilev D., Starikovskaya N. UVM based approaches to functional verification of communication controllers of microprocessor systems. In Proc. of the 2016 IEEE East-West Design & Test Symposium (EWDTS).

[2] PCI Express Base Specification Revision 3.0, http://pcisig.com/specifications

[3] Белянин И., Петраков П., Фельдман В. Функциональная организация и аппаратура сетевого взаимодействий модулей в вычислительном кластере на базе микропроцессоров с архитектурой «Эльбрус». Вопросы радиоэлектроники, серия ЭВТ, вып. 3, 2015 г., стр. 7-20.

[4] IEEE Standard for Ethernet. IEEE Std 802.3-2012. 1983 p.

[5] IEEE Standard for SystemVerilog — Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800-2012

[6] 1800.2-2017 - IEEE Standard for Universal Verification Methodology Language Reference Manual

[7] Стотланд И., Шпагилев Д., Петроченков М. Особенности функциональной веривикации контроллеров высокоскоростных каналов обмена микропроцессорных систем семейства «Эльбрус». Вопросы радиоэлектроники, серия ЭВТ, вып. 3, 2017, стр. 69-75.

[8] S. Chitti, P. Chandrasekhar, M. Asha Rani. Gigabit Ethernet Verification using Efficient Verification Methodology. In Proc. of the International Conference on Industrial Instruments and Control (ICIC), 2015, pp.1231-1235.

Программные решения для динамического изменения пользовательского интерфейса на основе автоматически собранной информации о пользователе

В.В. Зосимов <zosimovvv@gmail.com>
А.В. Христодоров <belfegor26@gmail.com>
А.С. Булгакова <sashabulgakova2@gmail.com>
Николаевский национальный университет им. В.А. Сухомлинского
Украина, 54030, г. Николаев, ул. Никольская, 24

Аннотация. В статье описываются функциональные возможности и структура программного модуля для автоматизированной адаптации интерфейсов веб-приложений. Особое внимание уделяется идентификации и различению псевдоанонимных пользователей веб-приложений для адаптации интерфейса под конкретного пользователя. Разработанный подход обеспечивает возможность псевдоидентификации киберсущностей в контексте поведения пользователей и автоматизированную адаптацию интерфейсов под особенности выделенного пользователя в зависимости от поставленных задач.

Ключевые слова: интерфейс; адаптивность; индуктивное моделирование; веб-интерфейс

DOI: 10.15514/ISPRAS-2018-30(3)-15

Для цитирования: Зосимов В.В., Христодоров А.В., Булгакова А.С. Программные решения для динамического изменения пользовательского интерфейса на основе автоматически собранной информации о пользователе. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 207-220. DOI: 10.15514/ISPRAS-2018-30(3)-15

1. Введение

Интерфейсы – это неотъемлемая составляющая звеньев восприятия программных продуктов и управления ими. Именно они обеспечивают управление программными продуктами и связь пользователей с программой. Неустанная интеграция IT-технологий в среду существования современного человека порождает все больший спрос на разработки в области адаптации

интерфейсов программных продуктов, направленных на удовлетворение потребностей пользователей.

Программные продуктов, ориентированные на повышение удобства пользования интерфейсами, обладают следующими достоинствами:

- упрощение восприятия бизнес-логики программного продукта конкретным пользователем;
- гибкость модели представления информации;
- повышение производительности работы с интерфейсом.

Под динамическим изменением интерфейса следует понимать его адаптацию в результате выполнения некоторого сценария на основе поведенческого портрета пользователя.

Наиболее полный обзор содержания адаптивного поведения человеко-машинного интерфейса (ЧМИ) содержится в [1]. Авторы предлагают три параметра интерфейса, которые могут меняться:

- содержание представляемой информации;
- форма представления информации и ведения диалога;
- распределение задач между человеком и машиной (уровень автоматизации).

В других работах декларируется, что адаптивность интерфейса проявляется:

- в настройке уровня детализации диалога с пользователем – от подробного диалога, «ведущего» пользователя к цели шаг за шагом через иерархию меню, к короткому, с использованием сокращенных команд и макросов в режиме «вопрос-ответ» [2];
- в подсказках, ограничении доступа к приложениям, регулировании интенсивности информационного обмена и изменению внешнего вида интерфейса [3,4];
- в фильтрации и расстановке приоритетов контента, предложенного пользователю; это может происходить не только в соответствии с собственными предпочтениями, но и под влиянием внешних факторов и контекста (например, пользователю предъявляется продукт, который выгодно купить именно сейчас [5]);
- в изменении темпа подачи информации [6];
- в настройке параметров изображения (толщины линий, размера шрифта, яркости и др.).

В условиях конкуренции современных разработок в области разработки и применения интерактивных интерфейсов повышается актуальность исследований в направлении автоматизированной адаптации пользовательских интерфейсов. Под автоматизированной адаптацией понимается динамическое изменение пользовательских интерфейсов на основе автоматически собранной информации о пользователе.

Значительную роль в исследовании моделей и методов адаптации интерфейсов играют достижения Питера Экерсли [7]. Отдельные вопросы проектирования и анализа использования возможностей псевдоидентификации рассматриваются в трудах исследователей Г. Кришна, К. Докинз, Дж. Гилмор [8, 9]. Некоторые вопросы использования технологий сбора данных в автономных киберфизических системах рассматриваются в работах отечественных ученых А. Бочкарева, В. Голембо [10].

В [8-10] рассмотрены фундаментальные принципы и концепции, которые применяются при реализации программных продуктов, представлены различные модели и методы, направленные на улучшение восприятия и взаимодействия с интерфейсами конечных программных продуктов. Но все эти аспекты рассматривались как отдельные задачи, и отсутствовало их объединение в единую адаптивную систему.

Анализ результатов предыдущих исследований позволил прийти к выводу, что на сегодняшний день неуклонно растет спрос на системы адаптации интерфейсов во всех сферах их использования, а работоспособные системы отсутствуют. В данной статье описывается разработанный авторами подход, доведенный до программной реализации.

1. Индуктивный подход к построению адаптивных интерфейсов

Пусть $CR = \{cr_1, \dots cr_{|CR|}\}$ – множество критериев, которые будут изменятся, то есть адаптироваться под пользователя, $Fact = \{fact_1, \dots fact_{|Fact|}\}$ – множество факторов, которые будут влиять на выбор того или иного критерия.

На основе выделенных заранее факторов, которые могут влиять на интерфейс, будут меняться критерии построения интерфейса. Индуктивность процесса заключается в том, что адаптация интерфейса происходит от конкретных данных наблюдений, то есть факторов – к общей модели [11], которая включает в себя множество критериев, которые будут меняться (например, при известной информации о возрасте пользователя (фактор –возраст), увеличить размер шрифта (для пожилых людей) – критерий.

Следовательно, исходя из вышесказанного, можно построить функцию Ф, отражающую процесс адаптации, где θ – параметры модели:

$$\Phi(cr_1(fact_1,\dots,fact_k),\dots,cr_m(fact_k,\dots,fact_F)) = \theta_0 + \sum_{i=1}^{m}\theta_i cr_i \sum_{F=1}^{k}(fact_1,\dots,fact_F) +$$

$$+ \sum_{i=1}^{m}\sum_{j=1}^{m}\theta_{ij} cr_i cr_j \sum_{F=1}^{k}(fact_1,\dots,fact_F) + \sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\theta_{ijk} cr_i cr_j cr_k \sum_{F=1}^{k}(fact_1,\dots,fact_F) + \dots$$

Общая функциональная схема механизма адаптации интерфейса представлена на рис.1. Механизм состоит из нескольких блоков: блок определения

факторов, где формируются/хранятся факторы пользователя, которые могут влиять на критерии изменения интерфейса, то есть адаптации; блок формирования критериев (количество критериев для каждого фактора может быть разной); блок обработки данных, в котором программным образом будет обрабатываться содержание критериев, приводит к изменению интерфейса [12].

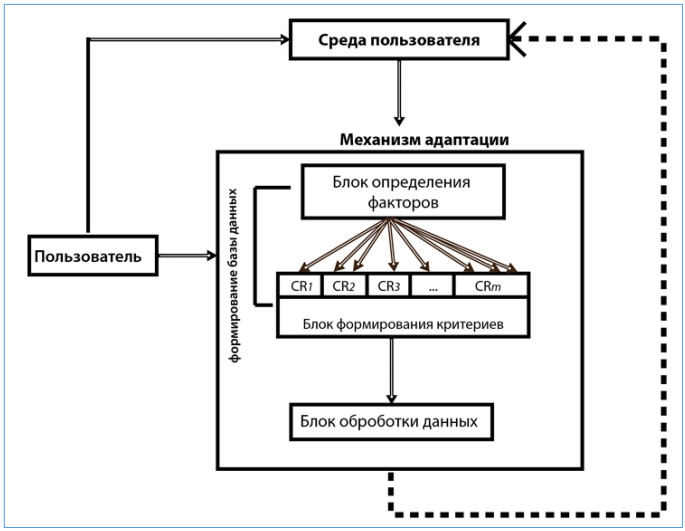


Рис.1. Составные части механизма адаптации интерфейса
Fig.1. Components of the interface adaptation mechanism

2. Требования к программному продукту

Программный продукт должен обеспечивать следующие функциональные возможности:

- сбор и хранение информации о пользователях соответствующего веб-приложения;
- псевдоидентификацию пользователей на основании собранных данных;
- автоматизированную адаптацию интерфейсов псевдоидентифицированных пользователей.

Программный продукт должен соответствовать следующим требованиям:

1. контролируемый сбор информации о пользователях веб-приложения;
2. создание и поддержка базы данных полученной информации;
3. псевдоидентификация пользователей веб-приложения на основе собранной информации;

4. автоматизированная адаптация пользовательского интерфейса веб-приложения на основании собранных и обработанных программным продуктом данных.

Ниже проведен сравнение предъявленных требований с возможностями существующих аналогов. За последние годы сформировалось и продолжает развиваться направление корректировки контента, структуры сайта, контроля внутренних и внешних факторов пребывания и взаимодействия пользователя с интерфейсом. Это неотъемлемая составляющая успеха маркетинговых и информационных сайтов. Существует достаточно большое количество систем сбора информации о пользователях, например, Amplitude [13], Mixpanel [14] и др. Эти программные продукты удовлетворяют требованиям 1 и 2, но не удовлетворяют требованиям 3 и 4. Практически отсутствуют системы, которые удовлетворяют всем четырем перечисленным требованиям.

3. Структурная схема программного продукта

На рис. 2 представлена схема взаимодействия программной системы «AAUI - Automatic Adaptation of User Interfaces».

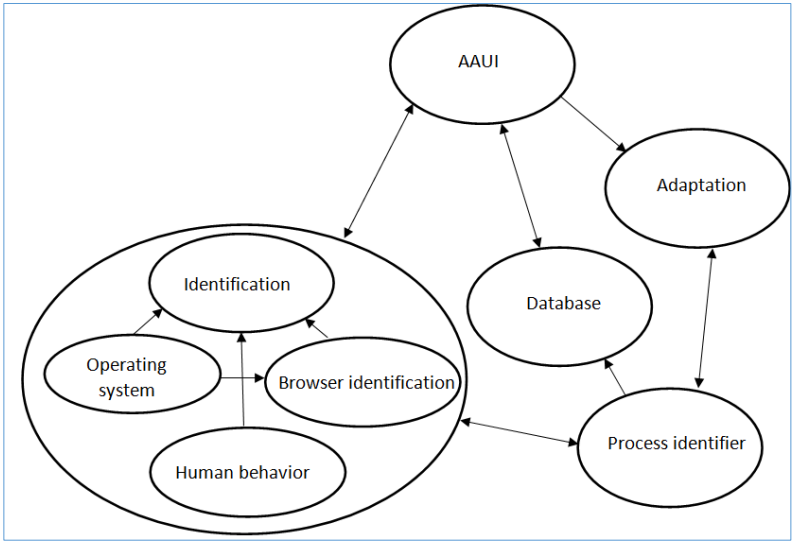


Рис 2. Схема взаимодействия программной системы «AAUI»
Fig. 2. Diagram of interaction of the "AAUI" software system

Блок «AAUI» является начальным модулем, начиная с которого происходит взаимодействие с компонентами программного продукта. «Identification» реализует формирование сущности на основании собранной информации об объекте, который проявил активность. «Operating system» обеспечивает сбор

информации о программных средствах активного объекта посещения. «Browser identification» отвечает за сбор информации о программном обеспечении, которое задействовано во взаимодействии с конечным программным продуктом. «Human behavior» обеспечивает проверку возможного прошлого присутствия псевдодеанимизированного объекта. «Adaptation» отвечает за программную адаптацию на основе созданного отпечатка псевдодеанимизированного объекта взаимодействия с конечным продуктом. «Database» отвечает за сохранность проанализированной информации, токенов псевдодеанимизации и правил адаптации интерфейсов. «Process identifier» отвечает за согласование идентификации существующих объектов взаимодействия.

На рис. 3 представлена блок-схема использования программного продукта.

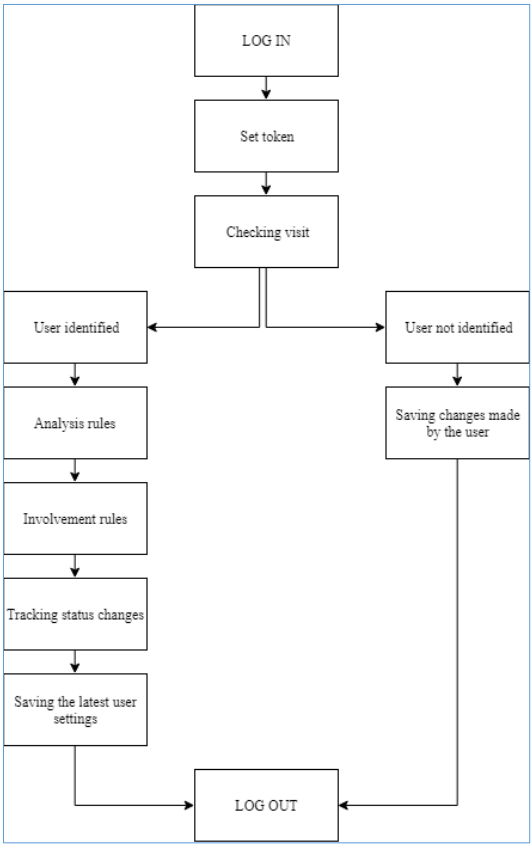


Рис 3. Схема использования программного продукта
Fig. 3. Scheme of using the software

В блоке «Set token» – программная система «AAUI» генерирует идентификатор. Генерация идентификатора происходит параллельно со сбором дополнительных данных для более точной идентификации пользователя, рис. 4.

```
function _getBrowserFingerprint() {
    $useragent = $_SERVER['HTTP_USER_AGENT'];
    $accept    = $_SERVER['HTTP_ACCEPT'];
    $charset   = $_SERVER['HTTP_ACCEPT_CHARSET'];
    $encoding  = $_SERVER['HTTP_ACCEPT_ENCODING'];
    $language  = $_SERVER['HTTP_ACCEPT_LANGUAGE'];
    $data = '';
    $data .= $useragent;
    $data .= $accept;
    $data .= $charset;
    $data .= $encoding;
    $data .= $language;

    $hash = hash('sha256', $data);
    return $hash;
}
```

Рис 4. Реализация блока генерации идентификатора
Fig. 4. Implementation of the ID generation unit

«Checking visit» – проверка на возможного предыдущего посещения сервиса пользователем.

- Если пользователь идентифицируется («User identified») происходит анализ поведенческих характеристик «Analysis rules», собранных на основе объектов интерфейса программного продукта, ориентированных на отслеживание изменений; применение персональных настроек «Involvement rules» и дальнейшее наблюдение за изменением потенциально изменяемых настроек интерфейса «Tracking status changes». Все изменения настроек веб-интерфейса, которые отслеживаются, сохраняются в базе данных каждый раз, когда они происходят – «Saving the latest user settings»;
- В случае, если пользователь впервые посещает веб-сервис, т.е. «User not identified», происходит отслеживание изменений, задаваемых пользователем, и их в базе данных сохранение «Saving changes made by the user».

4. Программная реализация

Основным преимуществом программной системы является примененная модель идентификации анонимных пользователей конечных программных продуктов с дальнейшим использованием динамического идентификатора для автоматической адаптации интерфейса под идентифицированного пользователя.

«AAUI» является серверным программным продуктом, написанным на скриптовом языке программирования PHP. Программный продукт обеспечивает:

- администрирование программного продукта (управление списком задействованных фильтров идентификации, изменение, редактирование, создание правил адаптации);
- хранение и обработка информации о пользователях;
- обеспечение хранения информации о сессиях.

В «AAUI» предусмотрен механизм настройки псевдодеанонимизации пользователей, и их группировки по заданным фильтрам. Функции групп пользователей:

1. группировка списков по странам посещения;
2. группировка списков пользователей по языку пользователей;
3. группировка по конфигурации конечного устройства посетителя;
4. комбинация групп для выделения целевой аудитории.

Надежность и устойчивое функционирование «AAUI» достигается совокупностью следующих организационно-технических мероприятий:

1. организация надежной защиты специалистами по кибербезопасности веб-приложений;
2. организации регулярного и качественного технического обслуживания серверной части;
3. своевременное обслуживание базы данных.

Серверное приложение «AAUI» взаимодействует с посетителем веб-приложения с начала его взаимодействия с интерфейсом веб-приложения (рис. 5). Уровень «поглощения» собранной информации о посетителе устанавливает администратор программной системы.

Рис 5. Схема модели взаимодействия
Fig. 5. Diagram of interaction model

Основным источником хранения данных является база данных. В ней хранится информация о полученных сущностях псевдодеанонимизированных пользователях. Для хранения информации используется свободная система управления реляционными базами данных MySQL. Структура базы данных разработана на основе разработанной методологии идентификации пользователей.

Для визуального представления схемы базы данных была использована методология IDEF1X, направленная на моделирование реляционных баз данных (рис.6). Указанный стандарт входит в семейство методологий IDEF позволяющее исследовать структуру, параметры и характеристики производственно-технических и организационно-экономических систем. Методология IDEF1X адаптирована для совместного использования с IDEF0 в рамках единой технологии моделирования. На основе IDEF0 детализируются функциональные блоки, а IDEF1X позволяет детализировать «стрелки». Разработка базы данных производилась с помощью инструмента для проектирования баз данных MySQL Workbench – инструмента, интегрирующего проектирование, моделирование, создание и эксплуатацию БД в единое окружение для системы баз данных MySQL с использованием, в частности, нотации IDEF1X.

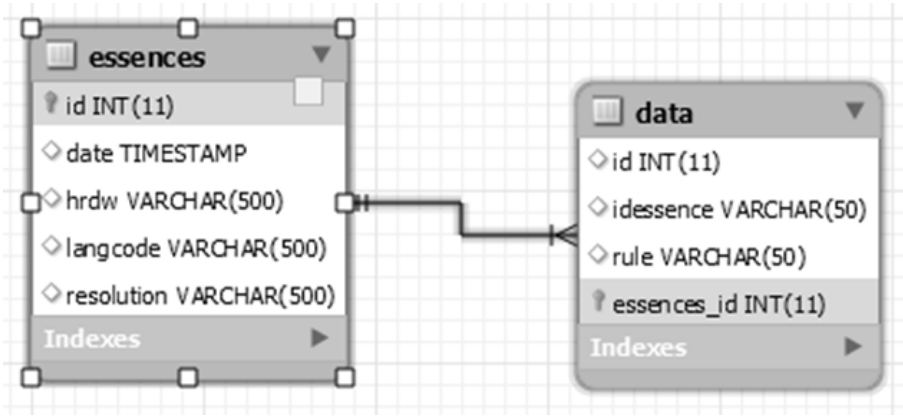


Рис 6. Фрагмент ER-диаграммы базы данных в нотации IDEF1X
Fig. 6. Fragment of the ER-diagram of the database in IDEF1X notation

Интересной в данном продукте является реализация системы псевдоидентификации пользователей. Взаимодействие частей программного продукта демонстрируют части кода, представленные на рис.7 и 8. На рис.7. представлен блок получения информации об установленных пользователем языков и часового пояса. Функция автоматической адаптации локализации веб-приложения на основе собранных данных представлена на рис.8.

5. Экспериментальное использование технологии

После интеграции «AAUI» с Интернет магазином было проведено исследование, направленное на выявление процентного соотношения прироста количества новых посетителей. В течение тридцати дней система работала в режиме сбора поведенческих сущностей, псевдодеанимизации и формирования базы посетителей.

Была получена выборка из 2000 уникальных поведенческих портретов. Из них 30% возвращались на сайт Интернет магазина в дальнейшем. На основании анализа собранных данных была запрограммирована функция адаптация, которая была более всего востребована – сортировка товаров по цене (от минимальной до максимальной).

```
App::before(function($request){
    $url_lang = Request::segment(1);
    $cookie_lang = Cookie::get('language');
    $browser_lang = substr(Request::server('HTTP_ACCEPT_LANGUAGE'), 0, 2);
    if(!empty($url_lang) AND in_array($url_lang, Config::get('app.languages')))
    {
        if($url_lang != $cookie_lang)
        {
            Session::put('language', $url_lang);
        }
        App::setLocale($url_lang);
    }
    else if(!empty($cookie_lang) AND in_array($cookie_lang, Config::get('app.languages')))
    {
        App::setLocale($cookie_lang);
    }

    else if(!empty($browser_lang) AND in_array($browser_lang, Config::get('app.languages')))
    {
        if($browser_lang != $cookie_lang)
        {
            Session::put('language', $browser_lang);
        }
        $timezone = \Auth::user()->timezone;

        $datetime = $this->asDateTime($value);
        DB::table('essences')->insert(
         ['langcode' => , $browser_lang) 'votes' => 0]
         ['date' => , $datetime + $timezone) 'votes' => 0]
        );
    }
    else
```

Рис 7. Блок сбора информации
Fig. 7. Information collection block

```
public function handle($request, Closure $next)
{
    if(!\Session::has('locale'))
    {
        \Session::put('locale', \Config::get('app.locale'));
    }
    $usersrule = DB::table('data')->select('rule', 'rule')->get();
    if(checkrule($usersrule) == Config::get('app.locale'))

    app()->setLocale(\Session::get('usersrule'));

    return $next($request);
}
```

Рис 8. Функция автоматической адаптации
Fig. 8. Automatic adaptation function

Программная система «AAUI», в течение тридцати дней генерировала выборку поведенческих портретов. При повторном посещении псевдодеанимизированного пользователя система автоматически адаптировала запрограммированную на отслеживание функцию в зависимости от полученного состояния сортировки.

В результате использования «AAUI» процент посетителей веб-приложения, которые снова обратились в Интернет-магазин, вырос с 30% до 50%.

В дальнейшем будет проведено более детальное экспериментальное исследование возможностей предлагаемой технологии.

6. Заключение

В работе представлено описание программного обеспечения для автоматизированной адаптации пользовательских интерфейсов. Проведенный анализ систем, направленных на сбор информации о пользователях, таких как Amplitude и Mixpanel, позволил сделать вывод, что указанные программные системы обеспечивают только сбор профилированной информации о входящих пользователях и не обеспечивают автоматизированной адаптации интерфейсов под нужды пользователей.

Разработанная система «AAUI» направлена на автоматизированную адаптацию интерфейсов под нужды пользователей. Система обеспечивает псевдоидентификацию пользователей (построение базы анонимных пользователей и правил на основе их пребывания в веб-приложениях). Оригинальной особенностью системы является использованная модель идентификации анонимных пользователей конечных программных продуктов с дальнейшим использованием динамического идентификатора для автоматической адаптации интерфейса под идентифицированного пользователя. В открытом доступе отсутствует аналогичное программное обеспечение.

К перспективам дальнейшей разработки «AAUI» можно отнести увеличение количества маркеров идентификации для повышения достоверности идентификации пользователей, внедрение в системы управления контентом и оптимизацию работы с данными.

Представлено экспериментальное применение технологии на примере Интернет-магазина, в котором в течение 30 дней было зафиксировано 2000 уникальных поведенческих портретов. В результате эксперимента процент посетителей веб-приложения, которые снова обратились в Интернет-магазин, вырос с 30% до 50%.

Список литературы

[1] Rothrock L., Koubek R., Fuchs F. et al. Review and reappraisal of adaptive interfaces: toward biologically inspired paradigms. Theoretical Issues in Ergonomics Science, vol. 3, No. 1, 2002, pp. 47-84.

[2] Ходаков В.Е., Ходаков Д.В. Адаптивный пользовательский интерфейс: проблемы построения. Автоматика. Автоматизация. Электротехнические комплексы и системы, № 1 (11), 2002, стр. 45-57.
[3] Курзанцева Л.И. Методика комплексного исследования адаптивного человеко-машинного интерфейса. Математические машины и системы, № 4, 2011, стр. 69-77.
[4] Курзанцева Л.И. Об адаптивном интеллектуальном интерфейсе «пользователь – система массового применения». Компьютерные средства, сети и системы, №7, 2008, стр. 110-116.
[5] Langley P. User modeling in adaptive interfaces. In Proceedings of the Seventh International Conference on User Modeling, 1999, pp. 357-370.
[6] Karwowski W. A review of human factors challenges of complex adaptive systems: discovering and understanding chaos in human performance. Human Factors, vol. 54, No. 6, 2012, pp. 983-995.
[7] Peter Eckersley, How Unique Is Your Web Browser? Electronic Frontier Foundation Режим доступа: https://www.eff.org/, дата обращения 02.05.2017.
[8] Gilmore J. Easy Laravel 5. Leanpub, 2016, 235 p.
[9] Dockins K. Design Patterns in PHP and Laravel. Appers, 2017, 45 p.
[10] О. Бочкарьов, В. Голембо. Використання інтелектуальних технологій збору даних у автономних кіберфізичних системах. Lviv Polytechnic National University Institutional Repository. Сборник научных трудов, № 830, 2015, стр. 7–11 (на украинском).
[11] Stepashko V., Bulgakova O., Zosimov V. Construction and research of the generalized iterative GMDH algorithm with active neurons. Advances in Intelligent Systems and Computing, vol. 689, 2018, pp. 492-510.
[12] Булгакова А.С. Концепция построения адаптивного интерфейса с использованием индуктивного подхода. Индуктивное моделирование сложных систем. Сборник трудов, выпуск 8. Киев: МННЦ ІТС, 2016, стр. 73-78 (на украинском)
[13] Amplitude. Analytics for modern product teams. Режим доступа: https://amplitude.com/, дата обращения 02.05.2017.
[14] Mixpanel. Product and User Analytics for Mobile, Web, and Beyond. Режим доступа: https://mixpanel.com/, дата обращения 02.05.2017.

Dynamically changing user interfaces: software solutions based on automatically collected user information

V.V. Zosimov <zosimovvv@gmail.com>
O.V. Khrystodorov <belfegor26@gmail.com>
O.S. Bulgakova <sashabulgakova2@gmail.com>
V.O. Sukhomlynsky Mykolaiv National University,
24 Nikolska St, Mykolayiv, 54030, Ukraine

Abstract. The developed system "AAUI" is aimed at the automated adaptation of interfaces to the needs of users. The system provides pseudo-identification of users (building anonymous users and rules database based on their stay in web applications). The original feature of the system is the used model of identifying anonymous users of the end products

with the further use of the dynamic identifier to automatically adapt the interface to the identified user's need. The analysis of the systems aimed at collecting information about users made it possible to conclude that these software systems provide only the collection of profiled information about incoming users and do not provide for automated adaptation of interfaces to the needs of users. There is no similar software in the open access. The prospects for the further development of AAUI include an increase in the number of identification markers to improve the authenticity of user identification, the integratioin with content management systems and the optimization of data management.

Keywords: interface, adaptiveness, inductive modelling, web-interface

DOI: 10.15514/ISPRAS-2018-30(3)-15

For citation: Zosimov V.V., Khrystodorov O.V., Bulgakova O.S. Dynamically changing user interfaces: software solutions based on automatically collected user information. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 207-220 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-15

References

[1] Rothrock L., Koubek R., Fuchs F. et al. Review and reappraisal of adaptive interfaces: toward biologically inspired paradigms. Theoretical Issues in Ergonomics Science, vol. 3, No. 1, 2002, pp. 47-84.

[2] Khodakov V.E., Khodakov D.V. Adaptive user interface: problems of building. Avtomatika. Avtomatizacija. Jelektrotehnicheskie kompleksy i sistemy [Automation. Automatization. Electrotechnical complexes and systems], № 1 (11), 2002, pp. 45-57 (in Russian).

[3] Kurgantseva L.I. Methodology of integrated studies of the adaptive human-machine interface. Matematicheskie mashiny i sistemy [Mathematical Machines and Systems], № 4, 2011, pp. 69-77 (in Russian).

[4] Kurgantseva L.I. About the adaptive intellectual interface “The user – system of mass application”. Komp'juternye sredstva, seti i sistemy [Computer means, networks and systems], №7, 2008, pp. 110-116 (in Russian).

[5] Langley P. User modeling in adaptive interfaces. In Proceedings of the Seventh International Conference on User Modeling, 1999, pp. 357-370.

[6] Karwowski W. A review of human factors challenges of complex adaptive systems: discovering and understanding chaos in human performance. Human Factors, vol. 54, No. 6, 2012, pp. 983-995.

[7] Peter Eckersley, How Unique Is Your Web Browser? Electronic Frontier Foundation Режим доступа: https://www.eff.org/, дата обращения 02.05.2017.

[8] Gilmore J. Easy Laravel 5. Leanpub, 2016, 235 p.

[9] Dockins K. Design Patterns in PHP and Laravel. Appers, 2017, 45 p.

[10] Botchkaryov A., Golembo V. Applying intelligent technologies of data acquisition to autonomous cyber-physical systems. Lviv Polytechnic National University Institutional Repository, № 830, 2015, стр. 7–11 (in Ukrain).

[11] Stepashko V., Bulgakova O., Zosimov V. Construction and research of the generalized iterative GMDH algorithm with active neurons. Advances in Intelligent Systems and Computing, vol. 689, 2018, pp. 492-510.

[12] Bulgakova O. The concept of constructing an adaptive interface using an inductive approach. Induktivnoe modelirovanie slozhnyh sistem [Inductive modeling of complex systems], pp. 73-78 (in Ukrain)
[13] Amplitude. Analytics for modern product teams. Available at: https://amplitude.com/, accessed 02.05.2017.
[14] Mixpanel. Product and User Analytics for Mobile, Web, and Beyond. Available at: https://mixpanel.com/, accessed 02.05.2017.

Variants of Chinese Postman Problems and a Way of Solving through Transformation into Vehicle Routing Problems

M.K. Gordenko <mgordenko@hse.ru>
S.M. Avdoshin <savdoshin@hse.ru>
Department of Software Engineering,
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. In this article, the routing problems are described. It is shown, that almost all routing problem can be transformed into each other. An example of the Mixed Chinese Postman problem is discussed. The article gives an overview of various variants of Chinese Postman Problem. For all problems the mathematical formulation is given. Moreover, the useful real-life application is presented, too. Then, the article provides a table of possible Chinese Postman problems and identifies parameters that can be varied for obtaining new problems. Five parameters have been identified, such as: presence of set of edges; presence of set of arcs; presence of edges with cost, depending on traversing; the presence of set of required edges; the presence of set of required arcs. It was shown that by varying these parameters one can obtain tasks that were not described earlier but can be used in real life. Four new tasks were identified. Then it is shown that the Chinese Postman problem can be solved as another routing tasks through graph transformations. The method for transforming Chinese Postman problem into the Generalized Travelling Salesman problem is given. Then the results of solving the above problem are presented by simple algorithms, and their effectiveness is shown. The research is not over yet. The testing of other algorithms is planned.

Keywords: Generalized Routing Problem; Arc Routing Problem; Chinese Postman Problem; Generalized Travelling Salesman Problem

DOI: 10.15514/ISPRAS-2018-30(3)-16

For citation: Gordenko M.K., Avdoshin S.M. The Variants of Chinese Postman Problems and Way of Solving through Transformation into Vehicle Routing Problems. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 221-232. DOI: 10.15514/ISPRAS-2018-30(3)-16

1. Introduction

The General Routing Problem (GRP) is a routing problem defined on a graph where a minimum cost tour is to be found and where the route must include visiting certain required vertices and traversing certain required edges [1]. The routing problems are

closely related to the logistic and transportation management. From the theoretical point of view, routing problems are mainly related to determining the optimal set of routes in a graph. In practice, the routing problems are not only the tasks of determining optimal set of routes, they are also the tasks of testing robots, the correctness of links in the application menu and operating systems, interactivity usability of web-sites [2]. The Travelling Salesman Problem (TSP) is one of the routing problem consisting in finding a minimal length closed tour that visits each city once. The TSP is one of the most well-known routing problem. Another practical, but less well-known problem is the Chinese Postman Problem (CPP). The CPP is finding a shortest closed path that visits every edge or arc of a graph. The CPP has a simple formulation and a lot of potentially useful applications, but today is poorly understood.

The article gives an overview of various CPPs, provides mathematical formulations of problems, and describes the scope of the problem. In addition, the article cites references to the literature, in which the various ways of transforming different types of ARP to VRP is described. Also, the results of the current research of various algorithms for solving the problem of a Generalized Traveling Salesman Problem (GTSP) are presented.

2. The Variations of Chinese Postman Problem

There are a lot of variations of CPP. Below, some of them are described.

2.1 The Windy Rural Chinese Postman problem

The Windy Rural Chinese Postman Problem (WRCPP) is a special case of ARP, in which $A_R \subseteq A$, $E_R \subseteq E$, and the cost of traversing the edges is depended from the direction of traversing.

WRCPP is a generalization of the CPP in a mixed multigraph. In original CPP problem, it is necessary to find a closed route of minimum length that contains all edges and arcs of the original multigraph at least once. In the real world, it is not always necessary to traverse absolutely all edges and arcs, it is enough to traverse only a certain set of them. Besides, the cost of traversing the edges depend from direction of traversing. The problem of this type is known as the Windy Rural Chinese Postman Problem, which is finding a closed route of minimum length that contains all required edges or arcs of the original multigraph at least once and can contains non-required edges or arcs, so, that the cost of traversing edges depends on traversing direction [5, 6].

Fix the edge $\{v_i, v_j\}$ (non-oriented pair of vertex) from E. Define (v_i, v_j) as ordered pair of vertices, meaning the traversing an edge $\{v_i, v_j\}$ from vertex v_i to v_j vertex. Note, that $(\exists\{v_i, v_j\} \in E)\big(C(v_i, v_j) \neq C(v_i, v_j)\big)$ (1)

Let arc be $(v_i, v_j) \in A$ ordered pair of vertices, meaning the traversing an arc (v_i, v_j) from vertex v_i to v_j vertex.

We give a formal formulation of the WRCPP problem, extending it to the case of a mixed multigraph.

Let $I = \{1,2,\ldots,|E_R + A_R|\}, L = \{1,2,\ldots,|V|\}$. On the set of vertices V of G define indexation $inv = V \to L, (\forall v_i \in V)(\forall v_j \in V)(v_i \neq v_j \to i \neq j), i = inv(v_i)$. On the set $E_R \cup A_R$ of G define indexation $inea = E_R \cup A_R \to I$, $(\forall u_i \in (E_R \cup A_R))(\forall u_j \in (E_R \cup A_R))\ (u_i \neq u_j \to i \neq j), i = inu(u_i)$.

The solution of WRCPP is the route $\mu = (v_{l_1}, u_{p_1}, v_{l_2}, u_{p_2}, \ldots, v_{l_k}, u_{p_k})$, which satisfy for the following [11]:

$$u_{p_i} = \begin{cases} (v_{l_i}, v_{l_{i+1}}), (v_{l_i}, v_{l_{i+1}}) \in E \\ (v_{l_i}, v_{l_{i+1}}), (v_{l_i}, v_{l_{i+1}}) \in A \end{cases} i = 1,2 \ldots, k-1$$

$$u_{p_k} = \begin{cases} (v_{l_k}, v_{l_1}), (v_{l_k}, v_{l_1}) \in E \\ (v_{l_k}, v_{l_1}), (v_{l_k}, v_{l_1}) \in A \end{cases} i = 1,2 \ldots, k-1$$

$E_R \cup A_R\{u_{p_1}, u_{p_2}, \ldots, u_{p_k}\} = \emptyset$

We denote by $C(\mu) = \sum_{i=1}^{k} C(u_{p_i})$ the cost of traversing the route.

Let M is a set of WRCPP routes. It is needed to find $\mu_0 \in M$, where $(\forall \mu \in M)\ (C(\mu 0) \leq C(\mu))$.

A lot of theoretical and computational works is devoted to WRCPP. WRCPP cannot be solved for polynomial time. In general, the problem of WRCPP is NP-hard [12].

2.2 The Undirected Rural Chinese Postman problem

The Undirected Rural Chinese Postman Problem (URCPP) is a particular WRCPP which consists of determining a minimum cost circuit on a graph so that it is possible to traverse a given subset of required edges.

DCPP is a special case of WRCPP, where $A = \emptyset$, and there is not edges, which satisfy (1). So, $\forall\{v_i,v_j\} \in E$, $C(v_i,v_j) = C(v_j,v_i)$.

The URCPP is known to be an NP-hard problem and it has some interesting real-life applications.

2.3 The Undirected Chinese Postman problem

The Chinese Postman problem in the undirected graph (Undirected Chinese Postman Problem, UCPP) is the original statement of the CPP problem, which was firstly introduced by the mathematician Kwang-Mei-Ko in 1960 [2].

UCPP is a special case of WRCPP, where $A = \emptyset$, $E_R = E$ and there is not edges, which satisfy (1). So, $\forall\{v_i,v_j\} \in E$, $C(v_i,v_j) = C(v_j,v_i)$

If multigraph has Eulerian circuit then this cycle is a solution of UCPP. The algorithm for constructing the Eulerian circuit has $O(|E|)$ time complexity [5].

The Eulerian circuit is existing in an undirected multigraph if multigraph is connected and every vertex has an even degree. A multigraph satisfying the conditions for the existence the Eulerian circuit is called Eulerian multigraph. If the original multigraph

is not Eulerian, then for UCPP solution some edges must be traversed more than once. In other words, the multigraph should be supplemented with copies of some the edges to the Eulerian multigraph, so that the cost of the added copies of the edges is minimal [4].

2.4 The Directed Rural Chinese Postman problem

The Directed Rural Chinese Postman Problem (DRCPP) is a special case of the WRCPP where a subset of the set of arcs of a given directed graph is required to be traversed at minimum cost [2, 8].

DRCPP is a special case of WRCPP, where $E = \emptyset$.

In general, the DRCPP is NP-hard for directed multigraphs.

This problem also known as the Selecting Chinese Postman problem [8].

2.5 The Directed Chinese Postman problem

The Chinese Postman problem in the directed graph (Directed Chinese Postman Problem, DCPP) is a special case of the WRCPP problem, in which defined on directed graph and all arcs should be traversed. In some articles DCPP also called New York Street Sweeper Problem [8].

DCPP is a special case of WRCPP, where $E = \emptyset$, $A_R = A$.

If multigraph has Eulerian trail, then this trail is a solution of DCPP. The algorithm for constructing the Eulerian trail has $O(|A|)$ time complexity [9].

The Eulerian trail is existing in a directed multigraph if multigraph is strongly connected and outdegree of each vertex is equal to indegree. A multigraph satisfying the conditions for the existence the Eulerian trail is called Eulerian multigraph. If the original multigraph is not Eulerian, then for DCPP solution some edges must be traversed more than once. In other words, the multigraph should be supplemented with copies of some the arcs to the Eulerian multigraph, so that the cost of the added copies of the arcs is minimal [2].

2.6 The Undirected Windy Rural Chinese Postman problem

The Undirected Windy Rural Chinese Postman Problem (UWRCPP) is an important ARP which generalizes most of the single-vehicle ARP and can be defined as follows [2, 9].

UWRCPP is a special case of WRCPP, where $A = \emptyset$ and there is edges, which satisfy (1).

2.7 The Undirected Windy Chinese Postman problem

The Undirected Windy Chinese Postman problem is the NP- hard problem of finding the minimum cost of a tour traversing all edges of an undirected multigraph, where the cost of traversal of an edge depends on the direction [10].

UWCPP is a special case of WRCPP, where $A = \emptyset$ and there is not edges, which satisfy (1). So, $\forall\{v_i,v_j\} \in E$, $C(v_i,v_j) = C(v_j,v_i)$.

If multigraph has Eulerian circuit then this cycle is a solution of WCPP. The algorithm for constructing the Eulerian circuit has $O(|E|)$ time complexity [5]. If the original multigraph is not Eulerian, then some should be traversed more than once. In other words, the multigraph should be supplemented with copies of the edges to the Eulerian multigraph so that the cost of the added copies of the edges is minimal. The solution of the complement problem for a graph that does not satisfy properties (9) and (10) is an NP-hard problem. Thus, WCPP belongs to the class of NP- hard that cannot be solved in polynomial time [13].

2.8 The Mixed Chinese Postman problem

Mixed Chinese Postman Problem (MCPP) it is a version of WRCPP, where multigraph consists from edges and arcs, simultaneously, and all of them should be traversed [11, 12].

MCPP is a special case of WRCPP, where $A_R = A$, $E_R = E$. and there is not edges, which satisfy (1). So, $\forall\{v_i, v_j\} \in E, C(v_i, v_j) = C(v_j, v_i)$.

In 1962, Ford and Fulkerson proposed necessary and sufficient conditions for a mixed graph to be Eulerian. It is necessary and sufficient that in a strongly connected multigraph, the degrees of all vertices are even, and the divergence of each vertex is zero. If a mixed multigraph does not satisfy these conditions, then it must be supplemented by copies of arcs and edges to the Eulerian multigraph, so that the cost of the added copies of the arcs and edges is minimal. The addition of a mixed multigraph to Eulerian is an NP-difficult problem [13].

2.9 The Mixed Windy Chinese Postman Problem

The Mixed Windy Chinese Postman Problem (MWCCP, also called WCPP) is a special case of WRCPP. In MWCCP the cost of traversing the edges is depended from the direction of traversing.

UWRCPP is a special case of WRCPP, where there are edges, which satisfy (1).

In many theoretical works it was shown that problem is NP- hard.

2.10 The Mixed Windy Chinese Postman Problem

The Mixed Rural Chinese Postman Problem (MRCCP) is a special case of WRCPP. In MRCCP not all edges and arcs should be traversed. There is a set of arcs and edges, which must appear in solution, other arcs and edges may appear in solution or may not.

MRCPP is a special case of WRCPP, where there are not edges, which satisfy (1).

In many theoretical works it was shown that problem is NP- hard [14].

We tried to build a classification of different CPP. Combine the existing CPP in a table containing the following criteria:

- the presence of set of edges (A),
- the presence of set of required edges (B),
- the presence of edges with cost, depending on traversing

- direction (C),
- the presence of set of arcs (D),
- the presence of set of required arcs (E).

The results are shown in "Table 1". As we can see, there are four problems, which today are not existing (yellow cells in table), but also can have real-world applications.

Table 1. The Classification of CPP

	UCPP	URCPP	UWCPP	UWRCPP	DCPP	DRCPP	MCPP	DRUCPP	MWCPP	DURWCPP	URDCPP	MRCPP	DRUWCPP	WRMCPP
A	-	-	-	-	+	+	+	+	+	+	+	+	+	+
B	-	-	-	-	-	+	-	-	-	-	+	+	+	+
C	-	+	-	+	-	-	-	+	-	+	-	+	-	+
D	+	+	+	+	-	-	+	+	+	+	+	+	+	+
E	-	-	+	+	-	-	-	-	+	+	-	-	+	+

3. Solving the Variations of Chinese Postman Problem

In many sources was shown that almost all ARP problems can be transformed into VRP problems, predominantly in generalized travelling salesman problems (GTSP) [13, 15, 16, 17]. For example, in [16] paper is described how the Capacitated Arc Routing Problem can be formulated as a standard vehicle routing problem. This allows us to transform arc routing into node routing problems and, therefore, establishes the equivalence of these two classes of problems. A well-known transformation by Pearn, Assad and Golden [16] reduces arc routing problem (ARP) into an equivalent vehicle routing problem (VRP). However, that transformation is regarded as unpractical, since an original instance with n required edges is turned into a VRP over a complete graph with $3n+1$ vertices. In [15] article was proposed a similar transformation that reduces this graph to $2n+1$ vertices, with the additional restriction that a previously known set of n pairwise disconnected edges must belong to every solution.

Thus, one can move from less studied problems ARP to well-known problems VRP, such as TSP and GTSP, which have a lot of different approximation algorithms for solving.

In the next sections, we try to compare the simplest algorithms for solving the GTSP.

Generalized travelling salesman problem (GTSP) is an expansion of well-known TSP (Travelling Salesman Problem). In GTSP all vertices of graph are grouped in separate clusters. The solution of GTSP is a minimum-cost route, which traverse each cluster exactly once.

4. Methods for Solving the Generalized Travelling Salesman Problem

Now, we investigate the following simple approximate algorithms for solving GTSP:

- Nearest Neighbor Heuristic (NN) [19];
- Repetitive Nearest Neighbor Heuristic (RNN) [20];
- Improved Nearest Neighbor Heuristic (INN) [21];
- Repetitive Improved Nearest Neighbor Heuristic (RINN) [22];
- Loneliest Neighbor Heuristic (NLN) [23].

To evaluate the developed algorithms, the source code was written in the C++language.

Experiments was conducted on Apple Macbook Pro 13 a1502. Measurements were made of the executing time of the algorithm and the error rate of the solution. The results is presented in Table 1, 2, 3, 4, 5 and 6. *Min(T), max(T)* and *M(T)* means minimum, maximum and average time of algorithm working. *Min(C), max(C)* and *M(C)* means minimum, maximum and average error rate of algorithms.

Table 2. The measurements of NN algorithms

|V|	min(T)	max(T)	$M(T)$	min(C)	max(C)	$M(C)$
50	0,001	0,079	0,003	6,65%	23,53%	14,59%
100	0,001	0,009	0,002	7,35%	21,25%	15,00%
200	0,002	0,025	0,004	8,93%	21,77%	15,24%
500	0,007	0,041	0,017	12,18%	35,04%	20,42%
1000	0,025	0,114	0,062	12,99%	40,77%	21,33%
1500	0,054	0,264	0,132	12,90%	37,38%	20,52%
2000	0,098	3,317	0,445	12,28%	39,08%	20,41%
3000	0,350	3,888	2,510	12,81%	42,00%	21,29%

Table 3. The measurements of RNN algorithms

|V|	min(T)	max(T)	$M(T)$	min(C)	max(C)	$M(C)$
50	0,011	0,088	0,033	4,31%	16,66%	9,84%
100	0,079	2,521	0,261	6,07%	15,52%	11,07%
200	0,630	3,365	1,504	6,72%	18,41%	12,51%
500	6,678	98,085	31,772	10,08%	30,40%	17,83%
1000	62,258	695,821	247,218	11,65%	37,44%	18,51%
1500	198,014	2009,900	763,293	11,40%	34,29%	18,68%
2000	489,153	6731,020	2224,092	11,29%	33,46%	18,67%
3000	4102,820	8901,420	6334,230	12,53%	38,94%	20,59%

Table 4. The measurements of INN algorithms

|V|	min(T)	max(T)	M(T)	min(C)	max(C)	M(C)
50	0,000	0,008	0,001	5,81%	25,32%	14,50%
100	0,000	0,009	0,001	7,48%	22,43%	14,79%
200	0,002	0,009	0,004	8,43%	22,36%	15,27%
500	0,011	0,054	0,025	12,57%	35,96%	20,30%
1000	0,040	0,178	0,095	12,64%	40,29%	20,63%
1500	0,091	0,395	0,207	12,24%	38,45%	20,31%
2000	0,155	1,035	0,404	12,09%	35,56%	20,24%
3000	0,369	21,215	4,478	12,65%	42,12%	21,20%

Table 5. The measurements of RINN algorithms

|V|	min(T)	max(T)	M(T)	min(C)	max(C)	M(C)
50	0,001	0,029	0,008	6,49%	22,54%	14,07%
100	0,004	0,652	0,067	7,17%	20,65%	14,86%
200	0,050	1,212	0,372	8,65%	23,28%	15,14%
500	0,018	0,178	0,072	12,57%	35,96%	20,30%
1000	0,094	1,177	0,446	12,64%	40,29%	20,63%
1500	0,212	6,003	1,769	12,24%	38,45%	20,36%
2000	0,556	28,860	6,816	12,09%	35,56%	20,24%
3000	1,007	114,590	28,321	12,65%	42,12%	21,20%

Table 6. The measurements of NLN algorithms

|V|	min(T)	max(T)	M(T)	min(C)	max(C)	M(C)
50	0,001	0,029	0,003	6,38%	22,21%	14,44%
100	0,004	0,046	0,011	8,03%	20,63%	14,76%
200	0,017	0,078	0,036	8,45%	21,76%	15,41%
500	0,078	0,434	0,190	12,52%	34,46%	20,36%
1000	0,293	2,262	0,814	12,65%	40,48%	20,68%
1500	0,656	15,192	3,043	12,81%	36,84%	20,17%
2000	0,356	19,953	3,764	12,59%	38,07%	21,32%
3000	1,456	120,110	27,402	13,34%	39,91%	22,33%

5. Summary

This article provides an overview of the known CPP. An attempt to systematize and classify these problems has been made. Mathematical formulations of new types of CPP was founded. The paper also shows that almost all problems of the ARP can be transformed to VRP. In addition, for solving the Chinese Postman problems the way of transformation it into VRP (mainly in GTSP) has been chosen.

At this stage, the research is not complete. It is necessary to investigate the various ways of transformation ARP is into VRP. In addition, it is necessary to investigate the various ways of solving the GTSP. And the key idea of future research is the use of transformation algorithms and algorithms for solving the GTSP for solving the different modifications of CPP.

References

[1]. Eglese R., Letchford A., General Routing Problem. In Encyclopedia of Optimization. Springer, Boston, MA. 2008.
[2]. Thimbleby, H. The directed chinese postman problem. Software: Practice and Experience, 33(11), 2003, pp. 1081-1096.
[3]. Toth P., Vigo D. (ed.). The vehicle routing problem. – Society for Industrial and Applied Mathematics, 2002.
[4]. Hertz A., Laporte G., Mittaz M. A tabu search heuristic for the capacitated arc routing problem. Operations research, vol. 48, no. 1, 2000, pp. 129-135.
[5]. Zerbino D. R., Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome research, vol. 18, no. 5, 2008, pp. 821-829.
[6]. Edmonds J., Johnson E. L. Matching, Euler tours and the Chinese postman. Mathematical programming, vol. 5, no. 1, 1973, pp. 88- 124.
[7]. Kolmogorov V. Blossom V: a new implementation of a minimum cost perfect matching algorithm. Mathematical Programming Computation, vol. 1, no. 1, 2009, pp. 43-67.
[8]. Robinson H. Graph theory techniques in model-based testing. In Proc. of the International Conference on Testing Computer Software, 1999.
[9]. Wilson R. J. An eulerian trail through Königsberg. Journal of graph theory, vol., 10, no. 3, 1986, pp. 265-275.
[10]. Ababei C., Kavasseri R. Efficient network reconfiguration using minimum cost maximum flow-based branch exchanges and random walks-based loss estimations, IEEE Transactions on Power Systems, vol. 26, no. 1, 2010, pp.. 30-37.
[11]. Chen W. H. Test sequence generation from the protocol data portion based on the Selecting Chinese Postman algorithm. Information Processing Letters, vol. 65, no. 5, pp. 261-268.
[12]. Aho A.V. et al. An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours. IEEE transactions on communications, vol. 39, no. 11, 1991, pp, 1604-1615.
[13]. Dror M. (ed.). Arc routing: theory, solutions and applications. Springer Science & Business Media, 2012.
[14]. Ghiani G., Improta G. An algorithm for the hierarchical Chinese postman problem. Operations Research Letters, vol. 26, no. 1, 2000, pp. 27- 32.
[15]. Longo H., De Aragão M. P., Uchoa E. Solving capacitated arc routing problems using a transformation to the CVRP. Computers & Operations Research, vol. 33, no. 6, pp. 1823-1837.
[16]. Pearn W. L., Assad A., Golden B. L. Transforming arc routing into node routing problems, Computers & operations research, vol. 14, no. 4, 1987, pp. 285-288.
[17]. Laporte G. Modeling and solving several classes of arc routing problems as traveling salesman problems. Computers & operations research, vol. 24, no. 11, 1997, pp. 1057-1061.

[18]. Fischetti M., Salazar González J. J., Toth P. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Operations Research, vol. 45, no. 3. 1997. pp. 378-394.
[19]. Solomon M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations research, vol. 35, no. 2, 1987, pp. 254-265.
[20]. Modares A., Somhom S., Enkawa T. A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. International Transactions in Operational Research, vol. 6, no. 6, 1999, pp. 591-606.
[21]. Cheung K.L., Fu A.W.C. Enhanced nearest neighbor search on the R-tree. ACM SIGMOD Record, vol. 27, no. 3, 1998. pp. 16-21.
[22]. Tao Y., Papadias D., Shen Q. Continuous nearest neighbor search. In Proceedings of the 28th International Conference on Very Large Databases, 2002, pp. 287-298.
[23]. Pimentel F. G. S. L. Double-ended nearest and loneliest neighbour: a nearest neighbour heuristic variation for the travelling salesman problem. Revista de Ciências da Computação, vol. 6, issue 6, 2011.

Варианты задач китайского почтальона и их решения через преобразование в задачи маршрутизации

М.К. Горденко <mgordenko@hse.ru>
С.М. Авдошин <savdoshin@hse.ru>
Департамент программной инженерии,
Национальный исследовательский университет "Высшая школа экономики",
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Abstract. В статье описаны проблемы маршрутизации. Показано, что почти все проблемы маршрутизации дуг могут быть преобразованы в другие проблемы маршрутизации. Это продемонстрировано на примере задачи китайского почтальона в смешанном мультиграфе. Также в статье приведен обзор различных задач китайского почтальона (в зависимости от типа графа, функции стоимости и обязательности прохождения элементов графа). Для каждой проблемы дана математическая постановка. Кроме того, описаны примеры потенциально полезных приложений, где задачи могут быть применены. Приведена таблица различных вариантов задачи китайского почтальона и выбраны параметры для идентификации различных типов задач. Выделено пять параметров: наличие дуг, наличие ребер, наличие обязательных дуг, наличие обязательных ребер, наличие ребер со стоимостью, зависящей от прохода. Показано, что, варьируя эти параметры, можно получить новые задачи, которые могут быть полезны в реальной жизни, однако еще не описаны. Выявлены четыре таких задачи. Показано, что задача китайского почтальона может быть решена путем преобразования в другие задачи маршрутизации. Приведен метод, позволяющий преобразовать задачу в обобщенную задачу коммивояжера. Показаны результаты применения простейших алгоритмов для решения преобразованного варианта задачи (результаты приведены для алгоритмов ближайшего соседа и их модификаций). Исследование еще не завершено, планируется продолжать тестировать алгоритмы решения смежных задач маршрутизации и алгоритмы для преобразований задач в эквивалентные.

Keywords: обобщенная задача коммивояжера; задача маршрутизации дуг; задача маршрутизации; обобщенная задача маршрутизации; задача китайского почтальона

DOI: 10.15514/ISPRAS-2018-30(3)-16

Для цитирования: Горденко М.К., Авдошин С.М. Варианты задач китайского почтальона и их решения через преобразование в задачи маршрутизации. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 221-232 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-16

Список литературы:

[1]. Eglese R., Letchford A., General Routing Problem. In Encyclopedia of Optimization. Springer, Boston, MA. 2008.
[2]. Thimbleby, H. The directed chinese postman problem. Software: Practice and Experience, 33(11), 2003, pp. 1081-1096.
[3]. Toth P., Vigo D. (ed.). The vehicle routing problem. – Society for Industrial and Applied Mathematics, 2002.
[4]. Hertz A., Laporte G., Mittaz M. A tabu search heuristic for the capacitated arc routing problem. Operations research, vol. 48, no. 1, 2000, pp. 129-135.
[5]. Zerbino D. R., Birney E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs. Genome research, vol. 18, no. 5, 2008, pp. 821-829.
[6]. Edmonds J., Johnson E. L. Matching, Euler tours and the Chinese postman. Mathematical programming, vol. 5, no. 1, 1973, pp. 88- 124.
[7]. Kolmogorov V. Blossom V: a new implementation of a minimum cost perfect matching algorithm. Mathematical Programming Computation, vol. 1, no. 1, 2009, pp. 43-67.
[8]. Robinson H. Graph theory techniques in model-based testing. In Proc. of the International Conference on Testing Computer Software, 1999.
[9]. Wilson R. J. An eulerian trail through Königsberg. Journal of graph theory, vol., 10, no. 3, 1986, pp. 265-275.
[10]. Ababei C., Kavasseri R. Efficient network reconfiguration using minimum cost maximum flow-based branch exchanges and random walks-based loss estimations, IEEE Transactions on Power Systems, vol. 26, no. 1, 2010, pp.. 30-37.
[11]. Chen W. H. Test sequence generation from the protocol data portion based on the Selecting Chinese Postman algorithm. Information Processing Letters, vol. 65, no. 5, pp. 261-268.
[12]. Aho A.V. et al. An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours. IEEE transactions on communications, vol. 39, no. 11, 1991, pp, 1604-1615.
[13]. Dror M. (ed.). Arc routing: theory, solutions and applications. Springer Science & Business Media, 2012.
[14]. Ghiani G., Improta G. An algorithm for the hierarchical Chinese postman problem. Operations Research Letters, vol. 26, no. 1, 2000, pp. 27- 32.
[15]. Longo H., De Aragão M. P., Uchoa E. Solving capacitated arc routing problems using a transformation to the CVRP. Computers & Operations Research, vol. 33, no. 6, pp. 1823-1837.
[16]. Pearn W. L., Assad A., Golden B. L. Transforming arc routing into node routing problems, Computers & operations research, vol. 14, no. 4, 1987, pp. 285-288.

[17]. Laporte G. Modeling and solving several classes of arc routing problems as traveling salesman problems. Computers & operations research, vol. 24, no. 11, 1997, pp. 1057-1061.

[18]. Fischetti M., Salazar González J. J., Toth P. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. Operations Research, vol. 45, no. 3. 1997. pp. 378-394.

[19]. Solomon M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints, Operations research, vol. 35, no. 2, 1987, pp. 254-265.

[20]. Modares A., Somhom S., Enkawa T. A self-organizing neural network approach for multiple traveling salesman and vehicle routing problems. International Transactions in Operational Research, vol. 6, no. 6, 1999, pp. 591-606.

[21]. Cheung K.L., Fu A.W.C. Enhanced nearest neighbor search on the R-tree. ACM SIGMOD Record, vol. 27, no. 3, 1998. pp. 16-21.

[22]. Tao Y., Papadias D., Shen Q. Continuous nearest neighbor search. In Proceedings of the 28th International Conference on Very Large Databases, 2002, pp. 287-298.

[23]. Pimentel F. G. S. L. Double-ended nearest and loneliest neighbour: a nearest neighbour heuristic variation for the travelling salesman problem. Revista de Ciências da Computação, vol. 6, issue 6, 2011.

Analysis of Mathematical Formulations of Capacitated Vehicle Routing Problem and Methods for their Solution

E. Beresneva <eberesneva@hse.ru>
S. Avdoshin <savdoshin@hse.ru>
Department of Software Engineering,
National Research University Higher School of Economics,
20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. Vehicle Routing Problem (VRP) is one of the most widely known questions in a class of combinatorial optimization problems. It is concerned with the optimal design of routes to be used by a fleet of vehicles to serve a set of customers. In this study we analyze Capacitated Vehicle Routing Problem (CVRP) – a subcase of VRP, where the vehicles have a limited capacity. CVRP is mostly aimed at savings in the global transportation costs. The problem is NP-hard, therefore heuristic algorithms which provide near-optimal polynomial-time solutions will be considered instead of the exact ones. The aim of this article is to make a survey on mathematical formulations of CVRP and on methods for solving each type of this problem. The first part presents a general information about the problem and restrictions of this work. In the second part, the classical mathematical formulations of CVRP are described. In the third part, a classification of most popular subcases of CVRP is given, including description of additional constraints with their math formulations. This section also includes most perspective methods that can be applied for solving special types of CVRP. The forth part contains an important note about the most powerful algorithm LKH-3. Finally, the fourth part consists of table with solving techniques for each subproblem and of scheme with basic problems of the CVRP class and their interconnections.

Keywords: capacitated vehicle routing problem; mathematical formulation; metaheuristics; classification of cvrp

DOI: 10.15514/ISPRAS-2018-30(3)-17

For citation: Beresneva E., Avdoshin S. Analysis of Mathematical Formulations of Capacitated Vehicle Routing Problem and Methods for their Solution. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018. pp. 233-250. DOI: 10.15514/ISPRAS-2018-30(3)-17

1. Introduction

The Vehicle Routing Problem (VRP) is one of the most widely known questions in a class of combinatorial optimization problems. VRP is directly related to Logistics

transportation problem and it is meant to be a generalization of the Travelling Salesman Problem (TSP). In contrast to TSP, VRP produces solutions containing some (usually, more than one) looped cycles, which are started and finished at the same point called "depot". The objective is to minimize the cost (time or distance) for all tours. For the identical type of input data, VRP has higher solving complexity than TSP. Both problems belong to the class of NP-hard tasks. Specialized algorithms are able to find optimal solutions for cases with up to about 50 customers; larger problems have been solved to optimality in some cases, but often at the expense of considerable computing time. Thus, actuality of research and development of heuristics algorithms for solving VRP is on its top, because such approximate algorithms can produce near-optimal solutions in a polynomial time. It is especially important in real-based tasks when there are more than one hundred clients in a delivery net.

Real world applications may be mail delivery, solid waste collection, street cleaning, distribution of commodities, design telecommunication, transportation networks, school bus routing, dial–a–ride systems, transportation of handicapped persons, and routing of sales people and maintenance units. A survey of real–world applications is in [1].

This work is aimed at analysis of VRP subcase, which is called Capacitated Vehicle Routing Problem (Capacitated VRP, CVRP), where the vehicles have a limited capacity. It means that there is a physical restriction on transportation more than determined amount of weight for each machine. Capacitated vehicle routing problems CVRP form the core of logistics planning and are hence of great practical and theoretical interest.

Nowadays, there is a great range of different variations of both classical mathematical model of CVRP and its subcases. It can be too difficult to understand all the details for newcomers in this field of study. It is important to have an ability not to waste personal time doing observation but to quickly get the best solution methods for the current problem. Unfortunately, there are no articles concerned with CVRP, which have both a full classification of the subcases and a list of the solving algorithms. So, the purpose of this study is to make a survey on subcases of CVRP and on state-of-the-art heuristic methods for solving each extension of this problem. Also, it was decided to provide a new variant of mathematical model differed from Integer Linear Programming models.

Clearly, a study of this type is inevitably restricted by various constraints, in this research only CVRP subcases with static and deterministic input are considered instead of the dynamic and stochastic ones. Another condition is that classification is based according to various types of constraints.

The paper is structured as follows. In the second part, the classical mathematical formulations of CVRP are described. In the third part, a classification of most popular subcases of CVRP is given, including description of additional constraints with their math formulations. This section also includes most perspective methods that can be applied for solving special types of CVRP. Finally, the fourth part consists of scheme with basic problems of the CVRP class and their interconnections and of conclusion.

2. CVRP mathematical model

In this paper, mathematical formulation of Asymmetrical CVRP (ACVRP) proposed by original authors [2] is adopted in a new way as follows. This new variant of math model is created because only Integer Programming models were found in other articles. ACVRP is chosen for basic formulation instead of Symmetrical CVRP (SCVRP) because the first one is a general variant of the second problem. In the paper we will use CVRP abbreviation having in mind the next formulation.

Given a complete weighted oriented graph $G = (V, A)$. Let $I = \{0, 1, \dots, N\}$, where $N = |V|$. Graph vertices are indexed as $= V \to I, (\forall v \in V)(\forall w \in V)\ v \neq w \Longrightarrow ind(v) \neq ind(w)$. Thus, $V = \{v_0, v_1, \dots, v_N\}$ is set of vertices, here $i = ind(v_i)$, and A is set of arcs. Let v_0 be a depot, where vehicles are located, and v_i be the destination points of a delivery, $i \neq 0$.

The distance between two vertices v_i and v_j is calculated using a distance function $c(v_i, v_j)$. Here a real-valued function $c(\cdot,\cdot)$ on $V \times V$ satisfies [3]:

- $c(v_i, v_j) \geq 0$ *(non-negativity axiom)*
- $c(v_i, v_j) = 0$ if and only if $v_i = v_j$ *(identity axiom)*

Each destination point v_i, $i = \overline{0..N}$, is associated with a known nonnegative demand, d_i, to be delivered, and the depot has a fictitious demand $d_0 = 0$. The total demand of the set $V' \subseteq V$ is calculated as $d(V') = \sum_{i \in |V'|} d_i$.

Let K be a number of available vehicles at the depot v_0. Each vehicle has the same capacity – C. Let us assume that every vehicle may perform at most one route and $K \geq K_{min}$, where K_{min} is a minimal number of vehicles needed to serve all the customers due to restriction on C. Clearly, next condition must be fulfilled – $(\forall v_i \in V)\ d_i \leq C$, which prohibits goods transportation that exceed maximum vehicle capacity.

Let introduce $V^0 = \{v_0\}$, where $v_0 \in V$. We divide V in $K + 1$ sets: $S = \{V^0, V^1, \dots, V^K\}$, each subset, except for V^0, represent a set of customers to be served for one vehicle. $S^{all} = \{S\}$ is a set of all possible partitions of V. Let $J = \{0, 1, \dots, K\}$ be a set that keeps indexes. Then $(\forall i \in J)\ |V^i| \geq 1$. There should be no duplicates in any of subsets from S: $(\forall i \in J)(\forall j \in J)$ $(i \neq j \Rightarrow V^i \cap V^j = \emptyset)$. Also, all subsets from S must form set V. Thus, $V = \bigcup_{i=0}^{K} V^i$. In this notation, we should make $V^{0i} = V^0 \cup V^i, i = \overline{1..K}$. It is obvious that $d(V^{0i}) \leq C, i = \overline{1..K}$.

Let introduce $M^i = \{1, \dots, N^i\}, N^i = |V^i|$, $\sum_{i=1}^{K} N^i = N$.So, $M^{0i} = \{0\} \cup M^i$. Let $I^i = \{i | i = ind(v), v \in V^i\}$ be a set of vertex indices from V^i. Then $I^{0i} = \{0\} \cup I^i$. Let $H^i = \{p^i \colon M^{0i} \to I^{0i} \mid p^i(0) = 0\ \&\ (\forall x \in M^{0i})(\forall y \in M^{0i})$ $x \neq y \Longrightarrow p^i(x) \neq p^i(y)\}$ be a set of codes of all Hamiltonian cycles $h^i = \left(v_{p^i(0)}, v_{p^i(1)}, \dots, v_{p^i(N^i)}\right)$ of V^{0i}. Weight of a Hamiltonian cycle $h^i \in H^i$ can be

found as $f(h^i) = c\left(v_{p^i(0)}, v_{p^i(N^i)}\right) + \sum_{j=0}^{N^i-1} c\left(v_{p^i(j)}, v_{p^i(j+1)}\right)$. Let S' be a set of $\{V^{01}, V^{02}, \dots, V^{0K}\}$. In this notation the weight of S' is calculated as $F(S') = \sum_{i=\overline{1..K}} f(h^i)$.

Overall, the formulation of CVRP is to find such $S^0: F(S^0) = \min_{S \in S^{all}} F(S)$.

If $c(v_i, v_j) = c(v_j, v_i)$ for $\forall v_i \in V \, \forall v_j \in V$ then the problem is symmetrical (SCVRP) and triangle inequality axiom must be hold $c(v_i, v_k) \leq c(v_i, v_j) + c(v_j, v_k)$.

According to [1], another variant of mathematical formulation of CVRP allows to leave some vehicles unused, it means that at most K circuits must be determined. Of course, the number of K_{min} must be less or equal than K.

In this case in the basic formulation described above we should subsequently divide V in $K' + 1$ sets: $S = \{V^0, V^1, \dots, V^K\}$, where $K' \in [K_{min}; K]$.

Alternative variant takes its place from real-based situations where available vehicles have their own capacity $C_i, i = \overline{1..K}$. Due to this fact, next restriction appears: $(V^{0i}) \leq C_i, \; i = \overline{1..K}$.

However, most researches put this alternative to another class of problems not connected with CVRP which is known as the Mixed Fleet VRP or as the Heterogeneous Fleet VRP. Thus, this variant will not be taken into consideration in this paper.

Among the best-known heuristic algorithms are those proposed by Pisinger and Ropke (2007) [4], Nagata and Braysy (2009) [5], and Vidal et al. (2012) [6].

3. Extensions of CVRP

3.1. Open VRP (OVRP)

The OVRP is a variant of the CVRP where the vehicles need not return to the depot after visiting the last customer of a given route. Any OVRP instance can be converted to an ACVRP instance by simply setting $c(v_i, v_0) = 0$.

There is only one heuristic algorithm for solving OVRP proposed by Salari et al. (2010) [9]. Their method is based on Integer Linear Programming Improvement Procedure.

There is a good variety of metaheuristics. Most known and important are following algorithms: Hybrid evolution strategy algorithm by Repoussis et al. (2010) [10], variant of Variable Neighborhood Search (VNS) algorithm for OVRP by Fleszar et al. (2009) [11], method based on Tabu Search (TS) with route-evaluations memories by Zachariadis and Kiranoudis (2010) [12], Yu et al. (2011) Genetic algorithm and the last one is Particle swarm optimization metaheuristic proposed by MirHassani and Abolghasemi (2011) [13].

3.2. Distance-Constrained CVRP (DCVRP)

The next extension of CVRP to be considered is Distance-Constrained CVRP [14]. It suggests introducing the maximum length or time constraint for each route. It means that the total travelled distance by each vehicle in the solution is less than or equal to the maximum possible travelled distance T. Thus, new function $t(v_i, v_j)$, returning travel time between v_i and v_j, appears.

Function $t(\cdot,\cdot)$ on $V \times V$ satisfies the same axioms as $c(\cdot,\cdot)$.

$$f^T(h^i) = t\left(v_{p^i(0)}, v_{p^i(N^i)}\right) + \sum_{j=0}^{N^i-1} t\left(v_{p^i(j)}, v_{p^i(j+1)}\right)$$

$$F^T(S') = \sum\nolimits_{i=\overline{1..K}} f^T(h^i)$$

$$(\forall i = \overline{1..K})(F^T(h^i) \le T_{Max})$$

Most heuristics applied to simple CVRP can be easily converted for solving DCVRP cases. However, one heuristic proposed by Li et al. stands out from them [15]. It transforms the DCVRP into a multiple traveling salesman problem with time windows.

3.3. VRP with Time Windows (VRPTW)

In VRPTW there is a constraint on time interval $[a_i; b_i]$ associated with each v_i, called time window. It means that service of each customer must start only after the time a_i comes and this service must end before the time b_i. Obviously, $a_0 = 0$ and $b_0 = \infty$ for v_0. Let us assume that if t_{cur} is a current time, then all vehicles leave v_0 when $t_{cur} = 0$. If a vehicle arrives to v_i at the moment when $t_{cur} < a_i$, then it is obliged to wait until $t_{cur} = a_i$ and to start serving only after that moment.

New function $t(v_i, v_j)$, returning travel time between v_i and v_j, appears. Also, a variable srv_i keeping serving time of v_i is introduced. It is clear, that the problem can be solved if $\{(\forall v_i \in V)(\exists v_j \in V) \mid a_i + srv_i + t(v_i, v_j) + srv_j \le b_j\}$.

There are a lot of metaheuristics for solving VRPTW, but the most actual and state-of-the-art ones are given. The guided Evolutionary algorithm of Repoussis et al. (2009) [16] combines evolution, ruin-and-recreate mutations and guided local search. Prescott-Gagnon et al. (2009) [17] suggests a Large Neighborhood search (LNS) combined with branch-and-price for solution reconstruction. The method proposed by Nagata et al. (2010) [18] uses an interesting relaxation scheme with penalized returns in time. Another algorithm (Vidal et al. (2013)) [19] also applies time-constraint relaxations during the search to benefit from infeasible solutions in the search space.

3.4. VRP with Backhauls (VRPB)

VRPB is another extension to CVRP. To define VRPB we need to divide the set of customers V^{oi} into two subsets: the first set contains customers who require the product to be delivered, these customers are called linehaul customers $L^i \subset V^{oi}$. The other set contains customers who require the product to picked up, they are called backhaul customers $B^i \subset V^{oi}$. $L^i \cap B^i = \emptyset, L^i \cup B^i = V^{oi}$. Also, neither all deliveries nor all pick-ups should exceed vehicle capacity: $d(L^i) \leq C$ & $d(B^i) \leq C$. If the tour contains customers from both sets, the linehaul customers must serve before any backhaul customers. Note that tours with backhaul customers only are not allowed in some formulations [1].

In basic formulation H^i shoud be changed as follows:

$$H^i = \{p^i: M^{oi} \to I^{0i} \big| p^i(0) = 0 \ \& \ (\forall x \in M^{oi})(\forall y \in M^{oi})$$

$$x \neq y \implies p^i(x) \neq p^i(y) \ \& \ ((x < y) \implies (v_{p^i(x)} \notin B^i) \ or \ (v_{p^i(y)} \notin L^i))\}$$

The best metaheuristics, according to [20], include the Adaptive LNS (ALNS) of Ropke and Pisinger (2006) [21], the Tabu Search (TS) of Zachariadis and Kiranoudis (2012) [22] which uses long-term memories to direct the search toward inadequately exploited characteristics; and finally multi-ant colony system algorithm by Gajpal and Abad (2009) [23], which suggests two multi-route local search schemes.

3.5. VRP with Backhauls and Time Windows (VRPBTW)

Like in VRPB, VRPBTW suggests having linehaul and backhaul customers. In addition, with every location v_i there is a service time srv_i associated for loading/unloading and a time window $[a_i; b_i]$, which specifies the time in which this service has to be provided. In the same way as for VRPTW, when arriving too early at a location v_i, i.e., before a_i, the vehicle is allowed to wait until a_i to start the service. Also, the linehaul customers must be served before any backhaul customers. Thus, mathematical formulation of VRPBTW is a combination of both formulations of VRPTW and VRP.

The most powerful algorithms for solving VRPBTW are those which are proposed by Thangiah et al. (1996) [24] and by Kucukoglu et al (2015) [25]. The first method is based on insertion procedure with improving through the application of λ-interchange and 2-opt exchange procedures. The second one includes combination of TS and SA.

3.6. VRP with Pickup and Delivery (VRPPD)

In the basic version of VRPPD, each customer v_i requests either two demands, d_i to be delivered and p_i to be picked up, or only $d_i = d_i - p_i$, that represents the difference between two demands. In addition, we need to add for each customer v_i two new variables: O_i which denotes the vertex where the source of delivery

originates and D_i which denotes the customer where the destination of the pick up exists. It should be noted that for each customer the delivery must be implemented before the pick up.

Let define $d(V_{part}^{0i}) - p(V_{part}^{0i}) \leq C, part = \overline{0..N^i}$, as the weight of the current load of the vehicle after visiting $v_{p^i(part)}$, where $d(V_{part}^{0i}) = \sum_{j \in |0..part|} d_j$, $part \leq N^i$ and $p(V_{part}^{0i}) = \sum_{j \in |0..part|} p_j$.

In basic formulation H^i shoud be changed as follows:

$$H^i = \{p^i \colon M^{oi} \to I^{0i} \big| \, p^i(0) = 0 \;\&\; (\forall x \in M^{0i})(\forall y \in M^{oi})$$
$$(x \neq y \implies p^i(x) \neq p^i(y)) \;\&\; (\forall x \in M^{oi})(\forall y \in M^{oi})$$
$$((v_{p^i(y)} = D_{p^i(x)}) \implies (x < y)) \;\&\; (\forall x \in M^{oi})(\forall y \in M^{oi})$$
$$((v_{p^i(x)} = O_{p^i(y)}) \implies (x > y))\}$$

A great number of heuristics and metaheuristics are presented in [26].

3.7. VRP with Simultaneous Pickup and Delivery (VRPSPD)

VRPSPD is a subcase of VRPPD where each customer is a linehaul as well as a backhaul customer. In VRPSPD each customer not only requires a given quantity of products to be delivered but also requires a given quantity of products to be picked up. A complete service (i.e., delivery and pickup) to the customer is provided by a vehicle in a single visit. Thus, there is no need to explicitely indicate both variables O_i and D_i as in VRPPD.

It is found in the literature that the heuristics of Subramanian et al. (2010) [27], Zachariadis and Kiranoudis (2011) [28] and Souza et al. (2010) [29] together produce the best known results.

3.8. VRP with Mixed Pickup and Delivery (VRPMPD)

VRPMPD is also a subcase of VRPPD where each customer has either a delivery demand or pickup. Therefore, there is $d_i > 0$ and $p_i = 0$ in the first case and $p_i > 0$ and $d_i = 0$. Nevertheless, in basic formulation H^i shoud be changed the same way as it was shown for VRPPD.

The best known heuristics are those of Subramanian (2013) which is based on Iterative Local Search (ILS) idea [30] and hybrid algorithm proposed by Subramanian, Uchoa and Ochi (2013) [31].

3.9. VRP with Pickup and Delivery and Time Windows (VRPPDTW)

The VRPPDTW in this paper contains all constraints in the VRPPD plus added constraints in which both pickup and delivery have given time windows. With every location v_i there is a service time srv_i associated for loading/unloading and a time window $[a_i; b_i]$, which specifies the time in which this service has to be provided. In

the same way as for VRPTW, when arriving too early at a location v_i, i.e., before a_i, the vehicle is allowed to wait until a_i to start the service. Also, for each customer the delivery must be implemented before the pick up.

Efficient neighborhood-centered metaheuristics have been proposed, including the ALNS of Ropke and Pisinger (2006) [32] and the two-phase method of Bent, and Van Hentenryck (2006) [33], which combines SA to reduce the number of routes with LNS to optimize the distance. However, these methods were recently outperformed by the memetic algorithm of Nagata and Kobayashi (2011) [34], which exploits a well-designed crossover focused on transmitting parent characteristics without introducing too many new arcs in the offspring.

3.10. Multi-depot VRP (MDVRP)

The MDVRP is a generalization of the CVRP where more than one depot may be considered. Obviously, the vehicle must start and end at the same depot.

So, part of basic formulation should be changed as follows:

Let G be a number of depots. Let introduce $V^0 = \{v_0^1, v_0^2, \dots, v_0^G\}$, where $v_0^i \in V$. In this case, we should make $V^{0i} = \{v_0^j \in V^0\} \cup V^i, i = \overline{1..K}, j = \overline{1..G}$.

The best heuristic approaches for the MDVRP are considered to be developed by Pisinger and Ropke (2006) [35] and Vidal et al. (2012) [6].

3.11. VRP with Multiple Use of Vehicles (VRPM) or Multi-Trip VRP (MTVRP)

VRPM or MTVRP is a variant of standard CVRP in which the same vehicle can be assigned to several routes during a given planning period. Not only this constraint is introduced but also the sum of the durations of the trips assigned to the same vehicle must not exceed T_{Max}. T_{Max} is a trip duration being the sum of the travel times on arcs used in the route. Thus, new function $t(v_i, v_j)$, returning travel time between v_i and v_j, appears.

In this variant it is possible if $d(V^{oi}) > C$, $i = \overline{1..K}$. We additionally divide V^i in MT_i sets: $V^i = \{V_1^i, V_2^i, \dots, V_{MT_i}^i\}$, where $MT_i \in [1; |V^i|]$. Let $J = \{0, 1, \dots, K\}$ be a set that keeps indexes. Then $(\forall i \in J)\ (\forall mt \in \overline{1..MT_i})\ |V_{mt}^i| \geq 1$. There should be no duplicates in any of subsets from V^i: $(\forall i \in J)\ (\forall mt_1 \in \overline{1..MT_i})\ (\forall mt_2 \in \overline{1..MT_i})\ (mt_1 \neq mt_2 \Rightarrow V_{mt_1}^i \cap V_{mt_2}^j = \emptyset)$.

Also, $V^i = \cup_{mt=1}^{MT_i} V_{mt}^i$. In this notation, we should make $(\forall mt \in \overline{1..MT_i}) V_{mt}^{0i} = V^0 \cup V_{mt}^i$. It is obvious that $(\forall mt \in \overline{1..MT_i})\ d(V_{mt}^{oi}) \leq C$.

Let introduce $M_{mt}^i = \{1, \dots, N_{mt}^i\}, N_{mt}^i = |V_{mt}^i|, \sum_{mt=1}^{MT_i} N_{mt}^i = N^i$. Then $M_{mt}^{0i} = \{0\} \cup M_{mt}^i$. Let $I_{mt}^i = \{i \mid i = ind(v), v \in V_{mt}^i\}$ be a set of vertex indices from V_{mt}^i. Then $I_{mt}^{0i} = \{0\} \cup I_{mt}^i$.

Let $H_{mt}^{i} = \left\{p_{mt}^{i}\colon M_{mt}^{0i} \to I_{mt}^{0i} \middle|\, p_{mt}^{i}(0) = 0 \;\&\; \left(\forall x \in M_{mt}^{0i}\right)\left(\forall y \in M_{mt}^{0i}\right) x \neq y \implies p_{mt}^{i}(x) \neq p_{mt}^{i}(y)\right\}$ be a set of codes of all Hamiltonian cycles $h_{mt}^{i} = \left(v_{p_{mt}^{i}(0)}, v_{p_{mt}^{i}(1)}, \dots, v_{p_{mt}^{i}(N^{i})}\right)$ of V_{mt}^{0i}.

Weight of a Hamiltonian cycle $h_{mt}^{i} \in H_{mt}^{i}$ can be found according to the formula:

$$f\left(h_{mt}^{i}\right) = c\left(v_{p_{mt}^{i}(0)}, v_{p_{mt}^{i}\left(N_{mt}^{i}\right)}\right) + \sum_{j=0}^{N_{mt}^{i}-1} c\left(v_{p_{mt}^{i}(j)}, v_{p_{mt}^{i}(j+1)}\right)$$

Let S' be a set of $\{V_{mt}^{01}, V_{mt}^{02}, \dots, V_{mt}^{0K}\}, mt = \overline{1..MT_i}$. In this notation the weight of S' is calculated as $F(S') = \sum_{i=\overline{1..K}} \sum_{mt=\overline{1..MT_i}} f(h_{mt}^{i})$.

Function $t(\cdot,\cdot)$ on $V \times V$ satisfies the same axioms as $c(\cdot,\cdot)$. And it is defined as $f^{T}\left(h_{mt}^{i}\right) = t\left(v_{p_{mt}^{i}(0)}, v_{p_{mt}^{i}\left(N_{mt}^{i}\right)}\right) + \sum_{j=0}^{N_{mt}^{i}-1} t\left(v_{p_{mt}^{i}(j)}, v_{p_{mt}^{i}(j+1)}\right)$.

The most important thing here is the next constraint:

$$\left(\forall i = \overline{1..K}\right) \sum_{mt=1}^{MT_i} f^{T}\left(h_{mt}^{i}\right) \leq T_{Max}$$

Overall, the formulation of VRPM is to find:

$$S^{0}\colon F(S^{0}) = \min_{S \in S^{all}} F(S)$$

Metaheuristic inspired by ideas of TS and adaptive memory-based search (AMS) (Taillard (1993) [36]) still shows good results. In addition, another variant of AMS by Olivera and Viera (2007) [37] is considered to be competitive.

3.12. Periodic VRP (PVRP)

The Periodic VRP (PVRP) is used when planning is done over a certain period and deliveries to the customer can be done in different days. For the PVRP, customers can be visited more than once, though often with limited frequency.

Efficient algorithm for solving PVRP is parallel extension of UTS with neighborhood-centered search (Cordeau and Maischberger, 2012 [38]). Also, the VNS of Hemmelmayr et al. (2009) [39], and the hybrid record-to-record and integer programming metaheuristic of Gulczynski et al. (2011) [40] can be successfully applied. In addition, one more metaheuristic is one that proposed by Vidal et al. (2012) [6]. It produces the current best solutions by combining the GA search breadth with efficient LS, relaxations schemes, and diversity management procedures.

3.13. Split Delivery VRP (SDVRP)

In the SDVRP-MDA, more than one vehicle can service a customer, so that a customer's demand can be split among several vehicles on different routes. The most important here is that split deliveries are allowed only if at least a minimum fraction of a customer's demand is delivered by each vehicle visiting the customer.

The first metaheuristic for a SDVRP is proposed in Chen et al. (2007) [41]. The idea of the approach is based on combination of the classical Clarke and Wright algorithm, the Mixed-Integer Linear Programming (MILP) model and variable length record-to-record travel methods. A similar procedure is applied in Gulczynski et al. (2010) [42] to the SDVRP with minimum delivery amounts, that is a SDVRP where each delivery to a customer should consist of at least a minimum amount of goods. Another metaheuristic which contains TS approach is proposed in 2008 by Archetti et al. [43] The main thing here is to obtain a reduced graph by removing some arcs and to apply a set covering MILP formulation for the best routes. And in Jin et al. (2008) [44] a set covering formulation is proposed and the problem is solved through column generation.

3.14. Cumulative CVRP (CCVRP)

CCVRP minimizes the sum of the arrival times at the customers instead of minimizing the total distance (or travel time) as an objective.

For the CCVRP, Ngueveu et al. (2010) [45] and Ribeiro and Laporte (2012) [46] modified the hybrid GA. Also, two-phase metaheuristic proposed by Ke and Feng in 2013 [47] is considered to be successful enough.

4. An important note

There is an important note about recent state-of-the-art algorithm proposed by K. Helsgaun [48]. In the end of 2017 this author released an extension of the Lin-Kernighan-Helsgaun TSP Solver for Vehicle Routing Problems, called LKH-3. In his technical report it is said that his algorithm can often obtain best known solutions for benchmark instances, and even new best solutions were found. Unfortunately, his algorithm cannot solve all subcases of CVRP. MDVRP, VRPM, PVRP, SDVRP and CCVRP can be solved using other metaheuristics but not using LKH-3.

5. Conclusions

Table 1 sums up abovementioned and shows a list of best metaheuristics for each defined subcase of CVRP.

The presented study is undertaken in order to make a survey on CVRP subcases and on heuristic methods for solving each extension of this problem. In addition, author variants of mathematical formulations are given.

Table 1. Best metaheuristics for CVRP subcases

#	Problem	Best metaheuristics
1	CVRP	LKH-3, Tabu Search, Simulated Annealing, Ant Colony Optimization algorithm, Genetic algorithm, Variable Neighborhood Search
2	OVRP	LKH-3, Evolution algorithm, Variable Neighborhood Search, Tabu Search, Genetic algorithm, Particle swarm optimization metaheuristic
3	DCVRP	LKH-3, CVRP + transformation to mTSP with Time Windows
4	VRPTW	LKH-3, Guided Evolutionary algorithm, Large Neighborhood search (LNS)
5	VRPB	LKH-3, Adaptive LNS, Tabu Search, Multi-Ant Colony System algorithm
6	VRPBTW	LKH-3, Tabu Search + Simulated Annealing, λ-interchange and 2-opt exchange procedures
7	VRPSPD	LKH-3, Genetic algorithm, Guided Evolutionary algorithm, Iterated Local Search algorithm
8	VRPMPD	LKH-3, Genetic algorithm, Guided Evolutionary algorithm, Iterated Local Search algorithm
9	VRPPDTW	LKH-3, Adaptive LNS, Simulated Annealing + LNS
10	MDVRP	Hybrid Genetic algorithm
11	VRPM or MTVRP	Adaptive Memory-Based Search variants
12	PVRP	Tabu Search, Variable Neighborhood Search, Genetic algorithm + LNS
13	SDVRP	Clarke and Wright Savings + Mixed-Integer Linear Programming, Tabu Search
14	CCVRP	Adaptive LNS, Variable Neighborhood Search

Fig. 1 sums up relations between classes of the CVRP and forms the classification of its subtypes. In our future work, we are going to extend current survey adding dynamic and stochastic subcases of CVRP.

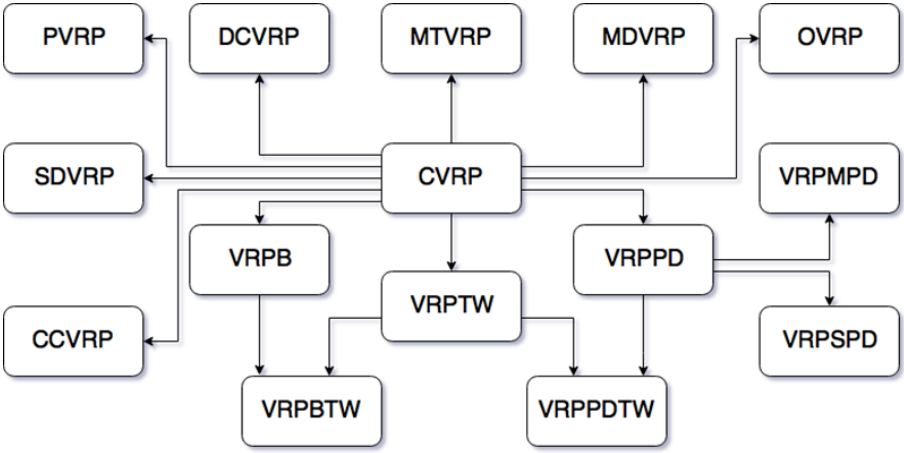


Fig. 1. The basic problems of the CVRP class and their interconnections

References

[1] P. Toth and D. Vigo. An overview of vehicle routing problems, In The Vehicle Routing Problem, SIAM, 2002.

[2] G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. Management Science, vol. 6, no. 1, 1959, pp. 80-91.

[3] M. Reed and B. Simon. Methods of Modern Mathematical Physics, London: Academic Press, 1972.

[4] B. Golden, S. Raghavan and E. Wasil. The vehicle routing problem: Latest advances and new challenges, New York: Springer, 2008.

[5] P. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. Computers & Operations Research, vol. 34, no. 8, 2007, pp. 2403-2435.

[6] Y. Nagata and O. Braysy. Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. Networks, vol. 54, no. 4, 2009, pp. 205-215.

[7] T. Vidal, T. Crainic, M. Gendreau, N. Lahrichi and W. Rei. A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. Operations Research, vol. 60, no. 3, 2012, pp. 611-624.

[8] M. Salari, P. Toth and A. Tramontani. An ILP improvement procedure for the Open Vehicle Routing Problem. Computers & Operations Research, vol. 37, no. 12, 2010, pp. 2106-2120.

[9] P. Repoussis, C. Tarantilis, O. Braysy and G. Ioannou. A hybrid evolution strategy for the open vehicle routing problem. Computers & Operations Research, vol. 37, no. 3, 2010, pp. 443-455.

[10] K. Fleszar, I. Osman and K. Hindi. A variable neighbourhood search algorithm for the open vehicle routing problem. European Journal of Operational Research, vol. 195, no. 3, 2009, pp. 803-809.

[11] E. Zachariadis and C. Kiranoudis. An open vehicle routing problem metaheuristic for examining wide solution neighborhoods. Computers & Operations Research, vol. 37, no. 4, 2010, pp. 712-723.

[12] S. MirHassani and N. Abolghasem. A particle swarm optimization algorithm for open vehicle routing problem. Expert Systems with Applications, vol. 38, no. 9, 2011, pp. 11547-11551.
[13] G. Laporte, Y. Nobert and M. Desrochers. Optimal routing under capacity and distance restrictions. Operations Research, vol. 33, no. 5, 1985, p. 1050–1073.
[14] C. Li, D. Simchi-Levi and M. Desrochers. On the distance constrained vehicle routing problem. Operational Research, vol. 40, 1992, pp. 790-799.
[15] P. Repoussis, C. Tarantilis and G. Ioannou. Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. IEEE Transactions on Evolutionary Computation, vol. 13, no. 3, 2009, pp. 624–647.
[16] E. Prescott-Gagnon, G. Desaulniers and L. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks, vol. 54, no. 4, 2009, pp. 190–204.
[17] Y. Nagata, O. Bräysy and W. Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research, vol. 37, no. 4, 2010, pp. 724–737.
[18] T. Vidal, T. Crainic, M. Gendreau and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers & Operations Research, vol. 40, no. 1, 2013, pp. 475–489.
[19] K. Braekers, K. Ramaekers and I. Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering, vol. 99, 2016, pp. 300-313.
[20] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research, vol. 171, no. 3, 2006, pp. 750–775.
[21] E. Zachariadis and C. Kiranoudis. An effective local search approach for the vehicle routing problem with backhauls. Expert Systems with Applications, vol. 39, no. 3, 2012, pp. 3174–3184.
[22] Y. Gajpal and P. Abad. Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. European Journal of Operational Research, vol. 196, no. 1, 2009, pp. 102–117.
[23] S. Thangiah, J.-Y. Potvin and T. Sun. Approaches to Vehicle Routing with Backhauls and Time. Windows International Journal of Computers and Operations Research, vol. 23, no. 11, 1996, pp. 1043-1057.
[24] I. Küçükoğlu and N. Öztürk. An advanced hybrid meta-heuristic algorithm for the vehicle routing problem with backhauls and time windows. Computers and Industrial Engineering, vol. 86, no. 3, 2015, pp. 60-68.
[25] S. Parragh, K. Doerner and R. Hartl. A survey on pickup and delivery problems. Journal für Betriebswirtschaft, vol. 58, 2008, pp. 81-117.
[26] A. Subramanian, Drummond, C. Bentes, L. Ochi and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Computers & Operations Research, vol. 37, no. 11, 2010, pp. 1899-1911.
[27] E. Zachariadis and C. Kiranoudis. A local search metaheuristic algorithm for the vehicle routing problem with simultaneous pick-ups and deliveries. Expert Systems with Applications, vol. 38, no. 3, 2011, pp. 2717-2726.
[28] M. Souza, M. Silva, M. Mine, L. Ochi and A. Subramanian. A hybrid heuristic, based on iterated local search and GENIUS, for the vehicle routing problem with simultaneous

pickup and delivery. International Journal of Logistics Systems Management, vol. 10, no. 2, 2010, pp. 142-156.
[29] A. Subramanian and M. Battarra. An iterated local search algorithm for the travelling salesman problem with pickups and deliveries. Journal of the Operational Research Society, vol. 64, 2013, pp. 402-409.
[30] A. Subramanian, E. Uchoa and L. Ochi. A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research, vol. 40, no. 10, 2013, pp. 2519-2531.
[31] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, vol. 40, no. 4, 2006, pp. 455–472.
[32] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research, vol. 33, no. 4, 2006, pp. 875–893.
[33] Y. Nagata and S. Kobayashi. A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. In Proceedings of PPSN'11, vol. 6238, 2011, pp. 536– 545.
[34] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. Computers & Operations Research, vol. 34, no. 8, 2007, pp. 2403–2435.
[35] E. Taillard, G. Laporte and M. Gendreau. Vehicle routeing with multiple use of vehicles. Journal of the Operational Research Society, vol. 47, no. 8, 1996, pp. 1065.
[36] A. Olivera and O. Viera. Adaptive memory programming for the vehicle routing problem with multiple trips. Computers & Operations Research, vol. 34, no. 1, 2007, pp. 28–47.
[37] J.-F. Cordeau and M. Maischberger. A Parallel Iterated Tabu Search Heuristic for Vehicle Routing Problems. Computers & Operations Research, vol. 39, no. 9, 2012, pp. 2033–2050.
[38] V. Hemmelmayr, K. Doerner and R. Hartl. A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research, vol. 195, no. 3, 2009, pp. 791–802.
[39] D. Gulczynski, B. Golden and E. Wasil. The period vehicle routing problem : New heuristics and real-world variants. Transportation Research Part E: Logistics and Transportation Review, vol. 47, no. 5, 2011, pp. 648-668.
[40] S. Chen, B. Golden and E. Wasil. The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. Networks, vol. 49, 2007, pp. 318–329.
[41] D. Gulczynski, B. Golden and E. Wasil. The split delivery vehicle routing problem with minimum delivery amounts. Transportation Research Part E, vol. 46, 2010, pp. 612–626.
[42] C. Archetti and M. Speranza. The split delivery vehicle routing problem: a survey. In The Vehicle Routing Problem Latest Advances and New Challenges, Operations Research, Computer Science Interfaces Series, 2008, pp. 103-122.
[43] M. Jin, K. Liu and B. Eksioglu. A column generation approach for the split delivery vehicle routing problem. Operations Research Letters, vol. 36, 2008, pp. 265-270.
[44] S. Ngueveu, C. Prins and C. Wolfler. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research , vol. 37, no. 11, 2010, pp. 1877–1885.
[45] G. Ribeiro and G. Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research, vol. 39, no. 3, 2012, pp. 728–735.

[46] L. Ke and Z. Feng. A two-phase metaheuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research, vol. 40, no. 2, 2013, pp. 633-638.
[47] J. Sze, S. Salhi and N. Wassan. The cumulative capacitated vehicle routing problem with min-sum and min-max objectives: An effective hybridisation of adaptive variable neighbourhood search and large neighbourhood search. Transportation Research Part B: Methodological, vol. 101, 2017, pp. 162-184.
[48] K. Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. Technical Report, Roskilde University, 2017.

Анализ математических постановок задачи маршрутизации с ограничением по грузоподъемности и методов их решения

Е.Н. Береснева <eberesneva@hse.ru>
С.М. Авдошин <savdoshin@hse.ru>
Департамент программной инженерии,
Национальный исследовательский университет "Высшая школа экономики",
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. Задача маршрутизации является одной из важнейших NP-трудных задач комбинаторной оптимизации. Она заключается в нахождении множества оптимальных замкнутых маршрутов с целью развозки товаров клиентам, используя ограниченное количество транспортных средств. В данной работе анализируется особый вид задачи маршрутизации – задача маршрутизации с ограничением по грузоподъемности, в которой у каждого транспортного средства есть лимит на максимальный вес (объем) груза. Целью данной работы является составление классификации различных типов задачи маршрутизации с ограничением по грузоподъемности. В первой части работы дана общая информация о проблеме, указаны рамки, в которых проводилось исследование – не рассматривались динамические и стохастические подвиды задачи маршрутизации. Во второй части представлена впервые предложенная авторами работы математическая постановка трех классических вариантов задачи маршрутизации с ограничением по грузоподъемности. В третьей части работы представлен список подклассов рассматриваемой задачи, включающий описание, математические модели для некоторых задач, а также наиболее перспективные метаэвристики, с помощью которых можно решать поставленную задачу. В четвертой части приведено упоминание об алгоритме LKH-3, способном решать некоторые подклассы задач с меньшим отклонением от оптимального значения по сравнению с другими алгоритмами. В заключении, приведена таблица, объединяющая все методы, описанные ранее, и схема с взаимосвязями задачи маршрутизации с ограничением по грузоподъемности и её подклассами. В будущем авторы работы планируют расширить классификацию, включив в неё подклассы стохастических и динамических вариантов данной проблемы.

Ключевые слова: задача маршрутизации с ограничением по грузоподъемности; математическая постановка; метаэвристики; классификация задач маршрутизации.

DOI: 10.15514/ISPRAS-2018-30(3)-17

Для цитирования: Береснева Е.Н., Авдошин С.М. Анализ математических постановок задачи маршрутизации с ограничением по грузоподъемности и методов их решения. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 233-250 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-17

Список литературы

[1] P. Toth and D. Vigo. An overview of vehicle routing problems, In The Vehicle Routing Problem, SIAM, 2002.
[2] G. B. Dantzig and J. H. Ramser. The Truck Dispatching Problem. Management Science, vol. 6, no. 1, 1959, pp. 80-91.
[3] M. Reed and B. Simon. Methods of Modern Mathematical Physics, London: Academic Press, 1972.
[4] B. Golden, S. Raghavan and E. Wasil. The vehicle routing problem: Latest advances and new challenges, New York: Springer, 2008.
[5] P. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. Computers & Operations Research, vol. 34, no. 8, 2007, pp. 2403-2435.
[6] Y. Nagata and O. Braysy. Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. Networks, vol. 54, no. 4, 2009, pp. 205-215.
[7] T. Vidal, T. Crainic, M. Gendreau, N. Lahrichi and W. Rei. A hybrid genetic algorithm for multi-depot and periodic vehicle routing problems. Operations Research, vol. 60, no. 3, 2012, pp. 611-624.
[8] M. Salari, P. Toth and A. Tramontani. An ILP improvement procedure for the Open Vehicle Routing Problem. Computers & Operations Research, vol. 37, no. 12, 2010, pp. 2106-2120.
[9] P. Repoussis, C. Tarantilis, O. Braysy and G. Ioannou. A hybrid evolution strategy for the open vehicle routing problem. Computers & Operations Research, vol. 37, no. 3, 2010, pp. 443-455.
[10] K. Fleszar, I. Osman and K. Hindi. A variable neighbourhood search algorithm for the open vehicle routing problem. European Journal of Operational Research, vol. 195, no. 3, 2009, pp. 803-809.
[11] E. Zachariadis and C. Kiranoudis. An open vehicle routing problem metaheuristic for examining wide solution neighborhoods. Computers & Operations Research, vol. 37, no. 4, 2010, pp. 712-723.
[12] S. MirHassani and N. Abolghasem. A particle swarm optimization algorithm for open vehicle routing problem. Expert Systems with Applications, vol. 38, no. 9, 2011, pp. 11547-11551.
[13] G. Laporte, Y. Nobert and M. Desrochers. Optimal routing under capacity and distance restrictions. Operations Research, vol. 33, no. 5, 1985, p. 1050–1073.
[14] C. Li, D. Simchi-Levi and M. Desrochers. On the distance constrained vehicle routing problem. Operational Research, vol. 40, 1992, pp. 790-799.
[15] P. Repoussis, C. Tarantilis and G. Ioannou. Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. IEEE Transactions on Evolutionary Computation, vol. 13, no. 3, 2009, pp. 624–647.
[16] E. Prescott-Gagnon, G. Desaulniers and L. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks, vol. 54, no. 4, 2009, pp. 190–204.

[17] Y. Nagata, O. Bräysy and W. Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. Computers & Operations Research, vol. 37, no. 4, 2010, pp. 724–737.

[18] T. Vidal, T. Crainic, M. Gendreau and C. Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. Computers & Operations Research, vol. 40, no. 1, 2013, pp. 475–489.

[19] K. Braekers, K. Ramaekers and I. Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. Computers & Industrial Engineering, vol. 99, 2016, pp. 300-313.

[20] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. European Journal of Operational Research, vol. 171, no. 3, 2006, pp. 750–775.

[21] E. Zachariadis and C. Kiranoudis. An effective local search approach for the vehicle routing problem with backhauls. Expert Systems with Applications, vol. 39, no. 3, 2012, pp. 3174–3184.

[22] Y. Gajpal and P. Abad. Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. European Journal of Operational Research, vol. 196, no. 1, 2009, pp. 102–117.

[23] S. Thangiah, J.-Y. Potvin and T. Sun. Approaches to Vehicle Routing with Backhauls and Time. Windows International Journal of Computers and Operations Research, vol. 23, no. 11, 1996, pp. 1043-1057.

[24] I. Küçükoğlu and N. Öztürk. An advanced hybrid meta-heuristic algorithm for the vehicle routing problem with backhauls and time windows. Computers and Industrial Engineering, vol. 86, no. 3, 2015, pp. 60-68.

[25] S. Parragh, K. Doerner and R. Hartl. A survey on pickup and delivery problems. Journal für Betriebswirtschaft, vol. 58, 2008, pp. 81-117.

[26] A. Subramanian, Drummond, C. Bentes, L. Ochi and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Computers & Operations Research, vol. 37, no. 11, 2010, pp. 1899-1911.

[27] E. Zachariadis and C. Kiranoudis. A local search metaheuristic algorithm for the vehicle routing problem with simultaneous pick-ups and deliveries. Expert Systems with Applications, vol. 38, no. 3, 2011, pp. 2717-2726.

[28] M. Souza, M. Silva, M. Mine, L. Ochi and A. Subramanian. A hybrid heuristic, based on iterated local search and GENIUS, for the vehicle routing problem with simultaneous pickup and delivery. International Journal of Logistics Systems Management, vol. 10, no. 2, 2010, pp. 142-156.

[29] A. Subramanian and M. Battarra. An iterated local search algorithm for the travelling salesman problem with pickups and deliveries. Journal of the Operational Research Society, vol. 64, 2013, pp. 402-409.

[30] A. Subramanian, E. Uchoa and L. Ochi. A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research, vol. 40, no. 10, 2013, pp. 2519-2531.

[31] S. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, vol. 40, no. 4, 2006, pp. 455–472.

[32] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers & Operations Research, vol. 33, no. 4, 2006, pp. 875–893.

[33] Y. Nagata and S. Kobayashi. A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. In Proceedings of PPSN'11, vol. 6238, 2011, pp. 536– 545.
[34] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. Computers & Operations Research, vol. 34, no. 8, 2007, pp. 2403–2435.
[35] E. Taillard, G. Laporte and M. Gendreau. Vehicle routeing with multiple use of vehicles. Journal of the Operational Research Society, vol. 47, no. 8, 1996, pp. 1065.
[36] A. Olivera and O. Viera. Adaptive memory programming for the vehicle routing problem with multiple trips. Computers & Operations Research, vol. 34, no. 1, 2007, pp. 28–47.
[37] J.-F. Cordeau and M. Maischberger. A Parallel Iterated Tabu Search Heuristic for Vehicle Routing Problems. Computers & Operations Research, vol. 39, no. 9, 2012, pp. 2033–2050.
[38] V. Hemmelmayr, K. Doerner and R. Hartl. A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research, vol. 195, no. 3, 2009, pp. 791–802.
[39] D. Gulczynski, B. Golden and E. Wasil. The period vehicle routing problem : New heuristics and real-world variants. Transportation Research Part E: Logistics and Transportation Review, vol. 47, no. 5, 2011, pp. 648-668.
[40] S. Chen, B. Golden and E. Wasil. The split delivery vehicle routing problem: Applications, algorithms, test problems, and computational results. Networks, vol. 49, 2007, pp. 318–329.
[41] D. Gulczynski, B. Golden and E. Wasil. The split delivery vehicle routing problem with minimum delivery amounts. Transportation Research Part E, vol. 46, 2010, pp. 612–626.
[42] C. Archetti and M. Speranza. The split delivery vehicle routing problem: a survey. In The Vehicle Routing Problem Latest Advances and New Challenges, Operations Research, Computer Science Interfaces Series, 2008, pp. 103-122.
[43] M. Jin, K. Liu and B. Eksioglu. A column generation approach for the split delivery vehicle routing problem. Operations Research Letters, vol. 36, 2008, pp. 265-270.
[44] S. Ngueveu, C. Prins and C. Wolfler. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. Computers & Operations Research , vol. 37, no. 11, 2010, pp. 1877–1885.
[45] G. Ribeiro and G. Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research, vol. 39, no. 3, 2012, pp. 728–735.
[46] L. Ke and Z. Feng. A two-phase metaheuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research, vol. 40, no. 2, 2013, pp. 633-638.
[47] J. Sze, S. Salhi and N. Wassan. The cumulative capacitated vehicle routing problem with min-sum and min-max objectives: An effective hybridisation of adaptive variable neighbourhood search and large neighbourhood search. Transportation Research Part B: Methodological, vol. 101, 2017, pp. 162-184.
[48] K. Helsgaun. An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems. Technical Report, Roskilde University, 2017.

Applying the methods of system analysis to teaching assistants' evaluation

E. Beresneva <eberesneva@hse.ru>
M. Gordenko <mgordenko@hse.ru>
Department of Software Engineering,
National Research University Higher School of Economics, 20, Myasnitskaya st., Moscow, 101000 Russia

Abstract. This article presents the results of applying various methods of system analysis (CATWOE, Rich Picture, AHP, Fuzzy AHP) to evaluation of teaching assistants. The soft and hard methods were applied. Methods of system analysis are considered as an example at the Higher School of Economics (HSE) in program "Teaching assistant". The article shows the process of interaction of teaching assistants with students and faculty in the form of Rich Picture. Selection and analysis of criteria for the evaluation of training assistants are carried out. Three groups of criteria were defined: professional skills, communicating skills, personal qualities. Each group has some subcriteria, which were defined in brainstorm. Its own method was determined, which immediately allow drop some assistants. In addition, the application of the methods AHP and Fuzzy AHP type-2 to evaluate teaching assistants is considered. The strengths and weaknesses of each method are revealed. It is also shown that, despite the power of the methods of system analysis, it is necessary to use common sense and logic. Do not rely only on the numbers obtained by the methods of system analysis. In the process of work, the best teaching assistant is selected, and the group of the best teaching assistants is defined.

Keywords: system analysis; combination of soft and hard methods; multicriteria decision making (MCDM); AHP; type-2 fuzzy sets; Fuzzy AHP

DOI: 10.15514/ISPRAS-2018-30(3)-18

For citation: Beresneva E., Gordenko M. Applying the methods of system analysis to teaching assistants' evaluation. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018. pp. 251-270. DOI: 10.15514/ISPRAS-2018-30(3)-18

1. Introduction

At the Higher School of Economics (HSE) there is a program "Teaching assistant" which has been effective for sever-al years. Each teacher can invite an education assistant, who will take some of the routine tasks related to teaching the course (checking homework, developing test materials, etc.).

Every student or a graduate student of the HSE, who meets the criteria established by the faculty, can be a teaching assistant. The teacher (or group of teachers) formulates tasks for the teaching assistants and monitors the quality of their performance. The teacher is responsible for the results of the students' knowledge, the quality of materials prepared by the education assistant, methodical support of the teaching assistant' work.

At the moment, all faculties establish their own criteria for selecting teaching assistants independently. Now there is only one criterion for all disciplines: "A student must have a mark at least 8 on the course in which he/she is involved, or he/she must have a recommendation from the department, to which teaching of this discipline is fixed." However, the practice shows that it is not enough to have only this criterion. There were no special studies about it before, but annual evidence showed that an excellent mark does not fully correlate with being a good teaching assistant. Recent year revealed that 60% of assistants were not able to cope with their work according to teachers. Most problems were connected with personal qualities, professional and communicative skills. For example, somebody did all the tasks slowly and did not do everything in time, or just did not have enough knowledge in the subject area. There were even some facts of disclosure of confidential information: one teaching assistant shared answers to the tests with students. Thus, there is a strong necessity to define a group of selective factors in a clever manner.

Recently, the head of Computer Science faculty has ordered each teacher (or group of teachers) on all disciplines to choose the best teaching assistant to give him/her an incentive award. In addition, next year the number of students is reduced, and it is necessary to decrease the number of assistants. Now there is a tendency on «Discrete mathematics» course that the education assistants who come from year to year are the same. This situation prompted the idea that at the moment when assessing teaching assistants, it is worth using additional criteria that will allow the group of teachers to select the best assistant and choose the group of the most successful assistants.

Thus, two tasks are faced – to choose the best assistant on «Discrete mathematics» course and to select the group of the most successful assistants, with whom it is possible to continue working on this course.

The purpose of this work is the development of searching method, which will select the best assistant and select the group of the most successful ones according to the criteria set by the group of teachers.

The rest of the paper is organized as follows. We discuss the problem specification in Section 2 and introduce our premises for model, which we use to illustrate our main results on Section 9. Sections 3, 4 and 6 present the different methods used for solution the problem. In sections 5 and 7 the derivations for the AHP and Fuzzy AHP are dis-cussed. Section 8 presents a sensitivity analyze.

2. The Difference between Previous Works and Our Approach

The literature review shows that there are a lot of researches that reveal a high success of applying the teaching assistant program in general. The most recent one is [3]. However, no one article is aimed neither at selection criteria for teaching assistants nor at searching methodology.

The closest study to our problem is devoted to a proposed framework for evaluating student's performance [4]. This work is based on the hard approach only. It uses the variation of the most widely used approach for multi-criteria decision-making – Analytic Hierarchy Process that combines mathematics and expert judgment. Since Analytic Hierarchy Process suffers from the problem of imprecision and subjectivity, their paper proposes to use Fuzzy AHP instead of traditional method.

However, there is an opinion about useless of applying Fuzzy AHP method. In [3] it is said that "the numerical representation of judgments in the AHP is already fuzzy" and "making fuzzy judgments more fuzzy does not lead to a better more valid outcome and it often leads to a worse one."

Our article proves that Fuzzy AHP with type-2 modification can still be used in a decision making process. Moreover, our study combines both hard and soft approaches be-cause this problem consists of not only main criteria but also it has a lot of additional ones. And these auxiliary factors can not be described using only formal algorithms.

3. Problem Definitions

The problem of finding the best teaching assistant and the group of teaching assistants is closely related with searching the criteria by which the teaching assistants should be selected.

To analyze the domain and determine its boundaries, the rich picture can be applied. Rich Picture is a collection of sketches, pictures, photos, symbols, signatures which represent a particular situation or a question of the real world from the point of view of the person or group of people who create it. Image components are people (stakeholders), systems, processes, inter-faces, data streams, information sources, infrastructure objects, attendant and impeding factors, emotions, points of view and attitude to them, etc.

Rich Picture can reflect the interaction and connections of the system components (or the surrounding world), their influence, cause and effect. It can also represent such subjective elements as attitude (perception), point of view, prejudice [1].

It is used to explore and aggregate the physical, conceptual and emotional aspects of the actual situation (sys-tem/problem/need).

Rich picture on subject «Teaching assistants» interactions in discipline «Discrete mathematic» is provided in Fig. 1.

To analyze the subject area and project boundaries, the CATWOE technique is a good addition to Rich Pictures.

CATWOE is defined by Peter Checkland as a part of his Soft Systems Methodology (SSM). It is a simple checklist for thinking. CATWOE is an acronym, each letter stands for a specific word: Clients, Actors, Transformation, World view, Owner, Environmental constraints [2].

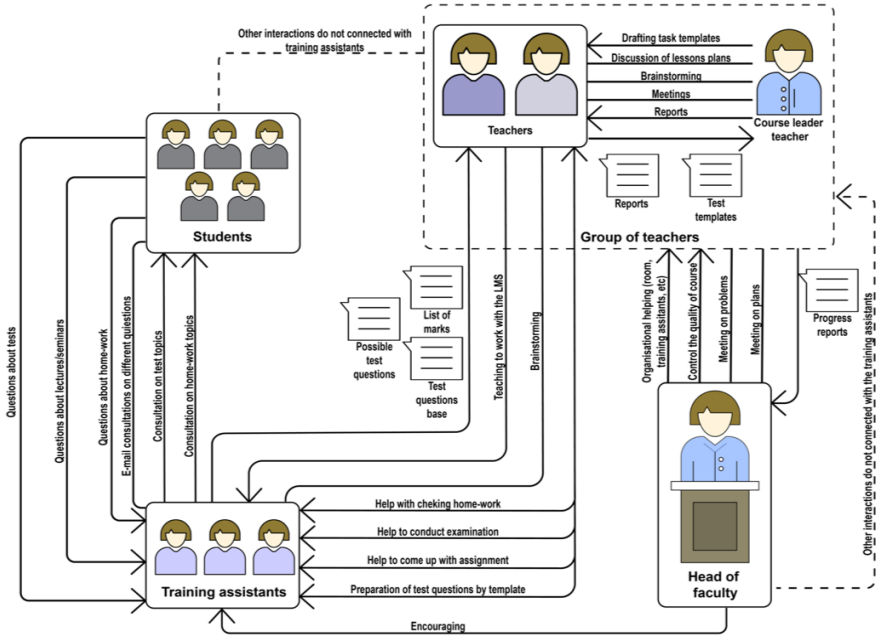


Fig. 1. Rich picture on subject "Teaching assistans' interactions in "Discrete mathematics"

Table 1. The CATWOE analysis

Role	Description
Clients	Teachers who want to assess their teaching assistants. Students who need assistants' help.
Actors	Groups of teachers who interested in evaluating of skills of teaching assistants and choosing the group of the best teaching assistants. The head of faculty who wants to encourage the best teaching assistant.
Transformation	Teaching assistant receives points for certain evaluation criteria.
World View	It is needed to define a group of the best teaching assistants and the best teaching assistant. The definition of a group of best teaching assistants is necessary in order to reduce the risks associated with incompetent and disinterested teaching assistants with the next year group of teaching assistants.
Owner	Teachers and the head of faculty.
Environmental	National educational and assessment standards.

constraints	

After analyzing the processes and interactions associated with the members of the system, a clear understanding of the subject area is emerged.

There are three teachers: one lecturer (the leading teacher) and two seminarians at "Discrete mathematics" course. They compose a decision group for choosing best assistants. Fair and reliable evaluation results would be obtained by this group because its members have a strong relationship with teaching assistants during the course.

In order to evaluate the assistants, it is decided to come up with evaluation criteria. After the first brainstorm, the list of criteria is similar to a chaotic list of records. The next meeting of the teachers shows that some of the criteria identified in the first stage for assessing the assistants turned out to be duplicated or unnecessary. After long discussions and joint brainstorming, three main groups of criteria are identified: professional skills, communicating skills, personal qualities.

The professional skills include the following sub-criteria:

- active involvement in the process of forming the program of discipline;
- initiative to compile new types of tasks for tests;
- knowledge of the subject domain;
- quality of homework checking;
- speed of homework cheking;
- experience of active use of the LMS;

The communicating skills include the following sub-criteria:

- pedagogical experience, the ability to correctly present information;
- openness to student issues (e.g. quick response to questions, competent answers);
- participation in counseling sessions before the tests and examinations;
- active communicating with teachers, participation in weekly meetings;
- the ability to listen carefully.

The personal qualities include the following sub-criteria:

- ethical compliance;
- punctuality;
- self-motivation, the desire for development;
- responsibility for work;
- teamwork skills;
- subordination;
- striving to achieve common results;
- resistance to conflict situations;
- the ability to generate new and innovative ideas;
- the ability to compromise;
- benevolence.

From the first group the next criteria are deleted:

- active involvement in the process of forming the program of discipline. *The teachers should do it, because drawing up a discipline program requires experience and entails a great responsibility;*
- knowledge of the subject domain. *Taking into account that each assistant is selected among the best students of the course, this requirement should be fulfilled by default.*

And the next criteria are combined as they characterize the checking of homework and are closely interrelated:

- quality of homework checking;
- speed of homework checking.

From the second group the next criteria are deleted:

- pedagogical experience, the ability to correctly present information. *This ability can be learned. One of the goals of the "Teaching Assistant" program is the development of teaching skills;*
- the ability to listen. *In our opinion, this parameter is almost impossible to estimate.*

From the third group the next criteria are combined, because they are very interrelated and cannot be separated:

- self-motivation, the desire for development;
- responsibility for work;

And the next criteria are deleted:

- teamwork skills. *It is related with the responsibility of work criteria;*
- ability to be subordinate. *By default, the main person on the course is the teacher. This is necessary to understand at first;*
- striving to achieve common results. *It is related with the responsibility of work criteria;*
- resistance to conflict situations. *It is the responsibility of the teacher to resolve and prevent the emergence of conflict situations;*
- the ability to generate new and innovative ideas. *This is not a paramount task of the teaching assistant. And the teaching assistant can work great, but do not come up with ideas, it's not scary;*
- the ability to compromise. *The last word for the teacher;*
- benevolence. *It is related with the ethical compliance and punctuality of work criteria.*

The final elected criteria and subcriteria are shown in Fig. 2. All the criteria and subcriteria have their own identification numbers.

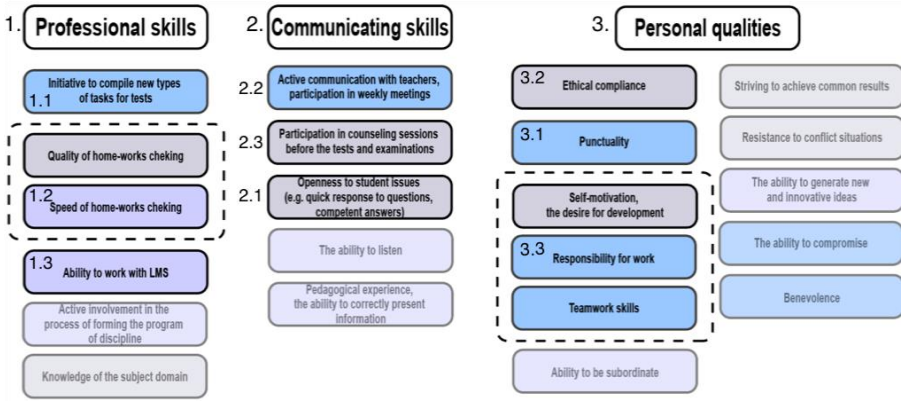


Fig. 2. The final list of criteria.

4. Exploring the alternatives

There are ten teaching assistants *A, B, C, D, E, F, G, H, I, J* on the course.

We can reduce the number of evaluating teaching assistants after assessing the involvement of teaching assistants in educational process.

We have 3 groups of criteria, consisting of 9 sub-criteria. In order to assess the involvement of assistants in the educational process, we did not use the values of the last three subcriteria (3.1-3.3). These sub-criteria refer to a group of personal qualities and cannot be regarded as involvement in the educational process. Then the involvement of the teaching assistant in the educational process for each criterion is evaluated, based on expert judgment. The results are presented in Table 2.

Table 2. The involvement in educational process

	A	B	C	D	E	F	G	H	I	J
1.1.	5	5	5	5	4	5	3	2	1	1
1.2.	4	5	5	4	4	4	4	4	1	4
1.3.	1	2	1	4	5	1	1	1	1	1
2.1.	4	3	3	4	2	5	5	1	3	1
2.2.	5	4	4	4	3	5	2	1	2	2
2.3.	4	4	3	4	1	4	5	2	1	2

Let us understand which assistants are least involved in the work process, according to experts. Calculations of threshold equals to 3, 4 and 5 are shown in Tables 3, 4 and 5.

Table 3.Treshhold is equal to 3

	A	B	C	D	E	F	G	H	I	J
1.1.	1	1	1	1	1	1	1	0	0	0
1.2.	1	1	1	1	1	1	1	1	0	1
1.3.	0	0	0	1	1	0	0	0	0	0
2.1.	1	1	1	1	0	1	1	0	1	0
2.2.	1	1	1	1	1	1	0	0	0	0
2.3.	1	1	1	1	0	1	1	0	0	0

Table 3 and 4 allow to identify teaching assistants who are least involved in the educational process.

The Table 5 with threshold equals to 5 shows that no one from H, I, J is not indispensable.

Table 4.Treshhold is equal to 4

	A	B	C	D	E	F	G	H	I	J
1.1.	1	1	1	1	1	1	0	0	0	0
1.2.	1	1	1	1	1	1	1	1	0	1
1.3.	0	0	0	1	1	0	0	0	0	0
2.1.	1	0	0	1	0	1	1	0	0	0
2.2.	1	1	1	1	0	1	0	0	0	0
2.3.	1	1	0	1	0	1	1	0	0	0

Table 5.Treshhold is equal to 5

	A	B	C	D	E	F	G	H	I	J
1.1.	1	1	1	1	0	1	0	0	0	0
1.2.	0	1	1	0	0	0	0	0	0	0
1.3.	0	0	0	0	1	0	0	0	0	0
2.1.	0	0	0	0	0	1	1	0	0	0
2.2.	1	0	0	0	0	1	0	0	0	0
2.3.	0	0	0	0	0	0	1	0	0	0

Thus, it is decided not to consider further the last three teaching assistants (H, I, J). However, little involvement in the educational process has its own explanations:

- H was ill two month;

- I was out of connection;
- J decided to switch to another faculty. Preparation for the exams took all his spare time.

Thus, seven candidates are remained. It is difficult to find the best one because each of them is successful in one or more criteria.

Stakeholders are about to choose A as a winner because this assistant took part in all teacher meetings and he suggested new types of tasks for tests so regularly (approximately once every two weeks). Assistant A communicated with teachers a lot (flashed before their eyes), that is why they prefer him.

However, this decision can be too unfair, that's why multicriteria decision making (MCDM) prosess is decided to be applied.

5. Analytical Hierarchy Process

Analytic Hierarchy Process (AHP) which is one of the most used MCDM approaches **Указан недопустимый источник.** is a structured multicriteria technique for organizing and analyzing complex decisions including many criteria. In this paper we use a classical AHP proposed by the author **Указан недопустимый источник.**.

At the first step of AHP a model for the decision is developed. Experts break down the decision into a hierarchy of goals, criteria, sub-criteria and alternatives.

After that, decisioners derive priorities (weights) for the criteria with respect to the desired goal. It is made in the form of pairwise comparisons using individual questionnaires. Since the evaluation criteria are subjective and qualitative in nature, it is very difficult for the decision maker to express the preferences using exact numerical values. That is why a special numerical scale **Указан недопустимый источник.** which consists of interpretation of linguistic terms is used (see Table 6).

Table 6. Saaty's pairwise comparison scale

Numeric value	**Linguistic terms**
1	Equally important
3	Moderately more important
5	Strongly more important
7	Very Strongly more important
9	Extremely important
2, 4, 6, 8	The intermediate values that are used to address situations of uncertainty between the two adjacent judgments

Results of comparisons of all experts are presented in the form of matrices (see Table 7).

Table 7. Criteria pairwise comparisons obtained by experts

	Professional skills			Communicative skills			Personal qualities		
	Exp.1	Exp.2	Exp.3	Exp.1	Exp.2	Exp.3	Exp.1	Exp.2	Exp.3

Professional skills	1	1	1	5	5	4	3	2	1
Communicative skills	1/5	1/5	1/4	1	1	1	1/3	1/4	1/5
Personal qualities	1/3	1/2	1	3	4	5	1	1	1

Before calculating the weights, the consistency of the comparison matrix is checked. As a rule, only if consistency is less than 0.1, it considered as acceptable, otherwise the pair-wise comparisons should be revised. In this decision making process, all of them are less than 0.092 that shows answers are consistence.

On the basis of Table 7 the final matrix is created by finding a mean between estimates of all experts (see Table 8). This metric is used because of solid decision to make all experts' voices to be equal.

Table 8. Aggregate matrix with criteria pairwise comparisons

	Professional skills	Communicative skills	Personal qualities
Professional skills	1	4,66	2
Communicative skills	0,22	1	0,25
Personal qualities	0,6	4	1

The matrix from Table 8 is used in order to calculate criteria priority weights. The same way as it was earlier, a pairwise comparison of all the sub-criteria, with respect to each criterion, included in the decision-making model, is made. Obtained results are shown in Table 9.

Table 9. Criteria and subcriteria priority weights

1. Professional skills	54,772%	1.1. New task types creation	25,232%
		1.2. HW checking	26,068%
		1.3. Experience in LMS	3,472%
2. Communicative skills	10,069%	2.1. Openness to students	2,946%
		2.2. Communication with teachers	1,288%
		2.3. Participation in consultations	5,834%
3. Personal qualities	35,159%	3.1. Punctuality	13,086%
		3.2. Ethical compliance	10,062%
		3.3. Self-motivation	12,011%

Next step consists of deriving the relative priorities (preferences) of the alternatives with respect to each criterion. Overall priority weights of assistants are calculated by summing all local priorities. Final figures are shown in Table 10. Bar chart is built on the basis of overall preferences of the alternatives (see Fig. 3).

Table 10. Local and overall priorities of alternatives

	A	B	C	D	E	F	G
1.1.	8,352%	6,711%	3,784%	2,728%	1,821%	1,214%	0,622%
1.2.	3,140%	6,060%	9,131%	3,836%	1,700%	0,966%	1,234%
1.3.	0,160%	0,249%	0,148%	1,001%	1,618%	0,148%	0,148%
2.1.	0,203%	0,154%	0,101%	0,441%	0,082%	0,803%	1,161%
2.2.	0,444%	0,089%	0,117%	0,062%	0,062%	0,444%	0,072%
2.3.	0,414%	0,387%	0,198%	0,833%	0,144%	1,628%	2,230%
3.1.	1,035%	2,677%	1,921%	3,670%	1,663%	1,041%	1,078%
3.2.	0,254%	1,928%	0,254%	3,822%	1,782%	0,993%	1,029%
3.3.	0,923%	2,040%	0,761%	4,201%	1,640%	1,108%	1,339%
Totals	**14,924%**	**20,296%**	**16,416%**	**20,595%**	**10,010%**	**8,846%**	**9,114%**

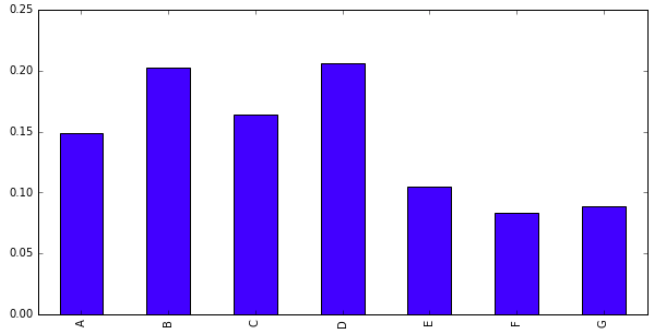


Fig. 3. Overall preferences of the alternatives

6. A Discussion on AHP Results

AHP analysis shows that the prompt decision of choosing A as the best assistant is totally unfair. Results reveal that experts did not take into account other important criteria that in general over weighted those, which were chosen at first. Another discovered problem of A is some of his/her estimates, which are the worst in comparison with others (for instance, criteria 3.1 and 3.2). This fact also decreases his/her chances to be a leader.

The main interesting point of results are the highest figures which belong to both two assistants B and D. Let's describe each of them.

Assistant B cannot be named as a brilliant employee. Nevertheless, he/she has showed good stable work without having bad results in any of the activities during the course. Despite not being the best in any of the criteria, B always was close to the leader.

In the same manner as B, assistant D has shown quite strong results in technical and communicative estimates. In addition, D was on the top in the personal qualities

criteria. He/she produces an impression of too self-motivated person and D was never late on any events. Result of D exceeds B at an inconspicuous gap of 0,3. Since experts make an arrangement on having no less than 2% advantage taking by the leader, such difference is admitted being not crucial for them.

In addition, there is a problematic situation with evaluation of the five best assistants. Four employees can be determined more or less clearly (A, B, C, and D). However, the difference between E and the closest competitor G is less than 1%, which is also insignificant.

7. Fuzzy Type-2 AHP

Since experts want to be more confident in fairness of their choice, we decide to apply another MCDM approach for purpose of aiming our goal. It is called Fuzzy AHP. In classical AHP crisp numbers are used, for pairwise comparison evaluations. However, in Fuzzy AHP, the linguistic variables are represented as fuzzy numbers instead of crisp. In this case a fuzzy logic provides a mathematical strength to capture the uncertainties associated with human cognitive process. Many researchers **Указан недопустимый источник.**, **Указан недопустимый источник.** who have studied the Fuzzy AHP have provided evidence that it shows relatively more sufficient description of decision making processes compared to the traditional AHP methods.

According to **Указан недопустимый источник.**, the membership functions of type-1 fuzzy sets have no uncertainty associated with it. Type-2 fuzzy sets generalize type-1 fuzzy sets and systems so that more uncertainty for defining membership functions can be handled. That's why type-2 fuzzy logic is used.

A type-2 fuzzy set $\tilde{\tilde{A}}$ in the universe of discourse X can be represented by a type-2 membership function $\mu_{\tilde{\tilde{A}}}$, shown as follows **Указан недопустимый источник.**:

$$\tilde{\tilde{A}} = \left\{ \left((x,u), \mu_{\tilde{\tilde{A}}}(x,u) \right) \middle| \forall x \in X, \forall u \in J_x \subseteq [0,1], 0 \le \mu_{\tilde{\tilde{A}}}(x,u) \le 1 \right\},$$

where J_x denotes an interval $[0,1]$.

$\tilde{\tilde{A}}$ is called an interval type-2 fuzzy set if all $\mu_{\tilde{\tilde{A}}} = 1$ **Указан недопустимый источник.**. Interval type-2 fuzzy sets are the most commonly used type-2 fuzzy sets because of their simplicity and reduced computational effort with respect to general type-2 fuzzy sets. For this reason, we use interval type-2 fuzzy sets.

A trapezoidal interval type-2 fuzzy set is illustrated as

$$\widetilde{\tilde{A}_i} = \left(\widetilde{A_i^U}; \widetilde{A_i^L} \right) = \left(a_{i1}^U, a_{i2}^U, a_{i3}^U, a_{i4}^U; H_1\left(\widetilde{A_i^U}\right), H_2\left(\widetilde{A_i^U}\right) \right), \left(a_{i1}^L, a_{i2}^L, a_{i3}^L, a_{i4}^L; H_1\left(\widetilde{A_i^L}\right), H_2\left(\widetilde{A_i^L}\right) \right),$$

where $\widetilde{A_i^U}$ and $\widetilde{A_i^L}$ are type-1 fuzzy sets, $a_{i1}^U, a_{i2}^U, \dots, a_{i3}^L, a_{i4}^L$ are the references points of the interval type-2 fuzzy set $\widetilde{\tilde{A}}_i$; $H_j\left(\widetilde{A_i^U}\right)$ denotes the membership value of the element $a_{j(j+1)}^U$ in the upper trapezoidal membership function $\widetilde{A_i^U}$ and $H_j\left(\widetilde{A_i^L}\right)$ denotes the membership value of the element $a_{j(j+1)}^L$ in the lower trapezoidal membership function $\widetilde{A_i^L}$, $j = \overline{1..2}$ **Указан недопустимый источник.**.

Pairwise comparison matrices got from experts for AHP are directly applied for our needs in Fuzzy AHP. We introduce interval trapezoidal type-2 fuzzy scales of the linguistic variables (see Table 11). They represent a modified version of scales proposed by **Указан недопустимый источник.** and include intermediate values between the two adjacent judgments like in AHP.

The priority weights of criteria (Table 12) and sub-criteria (Table 13) are demonstrated.

Type 2 fuzzy and defuzzified overall weights of the alternatives are shown in Tables 14 and 15. For defuzzification the Defuzzified Trapezoidal Type-2 Fuzzy Set (DTraT) approach is used proposed by **Указан недопустимый источник.**.

Bar chart is built on the basis of overall preferences of the alternatives (see Fig. 4).

Table 11. Trapezoidal interval type-2 fuzzy scales

Numeric value from AHP	Trapezoidal interval type-2 fuzzy scales
1	(1, 1, 1, 1; 1, 1) (1, 1, 1, 1; 1, 1)
2	(1, 1, 3, 4; 1, 1) (1.2, 1.2, 2.8, 3.8; 0.8, 0.8)
3	(1, 2, 4, 5; 1, 1) (1.2, 2.2, 3.8, 4.8; 0.8, 0.8)
4	(2, 3, 5, 6; 1, 1) (2.2, 3.2, 4.8, 5.8; 0.8, 0.8)
5	(3, 4, 6, 7; 1, 1) (3.2, 4.2, 5.8, 6.8; 0.8, 0.8)
6	(4, 5, 7, 8; 1, 1) (4.2, 5.2, 6.8, 7.8; 0.8, 0.8)
7	(5, 6, 8, 9; 1, 1) (5.2, 6.2, 7.8, 8.8; 0.8, 0.8)
8	(6, 7, 8.5, 9; 1, 1) (6.2, 7.2, 8.3, 8.8; 0.8, 0.8)
9	(7, 8, 9, 9; 1, 1) (7.2, 8.2, 8.8, 9; 0.8, 0.8)

Table 12. Interval type-2 fuzzy weights of criteria

Criteria	Interval type-2 weights
1. Professional skills	(0.275, 0.377, 0.754, 1.005; 1, 1) (0.304, 0.410, 0.706, 0.935; 0.8, 0.8)
2. Communicative skills	(0.057, 0.073, 0.14, 0.211; 1, 1) (0.061, 0.078, 0.13, 0.19; 0.8; 0.8)
3. Personal qualities	(0.188, 0.257, 0.519, 0.708; 1, 1) (0.203, 0.274, 0.477, 0.639; 0.8, 0.8)

Table 13. Interval type-2 fuzzy weights of sub-criteria

Sub-criteria	Interval type-2 weights
1.1.	(0.071, 0.134, 0.447, 0.811; 1, 1) (0.085, 0.154, 0.396, 0.703; 0.8, 0.8)
1.2.	(0.074, 0.138, 0.453, 0.811; 1, 1) (0.088, 0.158, 0.402, 0.705; 0.8, 0.8)
1.3.	(0.013, 0.022, 0.069, 0.124; 1, 1) (0.015, 0.025, 0.061, 0.108; 0.8, 0.8)
2.1.	(0.008, 0.014, 0.062, 0.149; 1, 1) (0.009, 0.016, 0.052, 0.118; 0.8, 0.8)

2.2.	(0.004, 0.007, 0.031, 0.071; 1, 1) (0.004, 0.008, 0.025, 0.055; 0.8, 0.8)
2.3.	(0.014, 0.029, 0.115, 0.24; 1, 1) (0.017, 0.034, 0.099, 0.199; 0.8, 0.8)
3.1.	(0.055, 0.087, 0.212, 0.317; 1, 1) (0.062, 0.095, 0.191, 0.28; 0.8, 0.8)
3.2.	(0.046, 0.069, 0.168, 0.265; 1, 1) (0.051, 0.075, 0.151, 0.23; 0.8, 0.8)
3.3.	(0.046, 0.075, 0.196, 0.317; 1, 1) (0.052, 0.082, 0.175, 0.274; 0.8, 0.8)

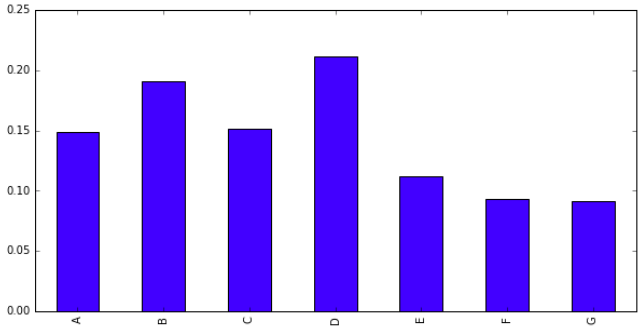


Fig. 4. Overall preferences of the alternatives

Table 14. Local and overall priorities of alternatives A, B, C, D

	A	B	C	D
1.1.	(0.115, 0.212, 0.505, 0.806; 1, 1) (0.133, 0.234, 0.462, 0.7; 0.8, 0.8)	(0.098, 0.174, 0.414, 0.692; 1, 1) (0.113, 0.191, 0.38, 0.612; 0.8, 0.8)	(0.048, 0.091, 0.252, 0.484; 1, 1) (0.06, 0.1, 0.23, 0.4; 0.8, 0.8)	(0.037, 0.063, 0.173, 0.342; 1, 1) (0.042, 0.07, 0.15, 0.3; 0.8, 0.8)
1.2.	(0.04, 0.067, 0.158, 0.272; 1, 1) (0.045, 0.073, 0.14, 0.24; 0.8, 0.8)	(0.083, 0.144, 0.314, 0.471; 1, 1) (0.095, 0.157, 0.3, 0.429; 0.8, 0.8)	(0.186, 0.265, 0.477, 0.657; 1, 1) (0.202, 0.283, 0.45, 0.6; 0.8, 0.8)	(0.067, 0.108, 0.26, 0.406; 1, 1) (0.077, 0.12, 0.24, 0.37; 0.8, 0.8)
1.3.	(0.038, 0.042, 0.055, 0.067; 1, 1) (0.039, 0.043, 0.05, 0.064; 0.8, 0.8)	(0.038, 0.051, 0.082, 0.106; 1, 1) (0.041, 0.054, 0.08, 0.1; 0.8, 0.8)	(0.032, 0.037, 0.051, 0.067; 1, 1) (0.033, 0.04, 0.05, 0.063; 0.8, 0.8)	(0.177, 0.229, 0.382, 0.536; 1, 1) (0.187, 0.24, 0.36, 0.5; 0.8, 0.8)
2.1.	(0.029, 0.044, 0.086, 0.133; 1, 1) (0.032, 0.047, 0.08, 0.12; 0.8, 0.8)	(0.027, 0.038, 0.072, 0.11; 1, 1) (0.029, 0.04, 0.07, 0.1; 0.8, 0.8)	(0.021, 0.026, 0.046, 0.071; 1, 1) (0.022, 0.03, 0.043, 0.06; 0.8, 0.8)	(0.064, 0.104, 0.237, 0.409; 1, 1) (0.072, 0.113, 0.22, 0.36; 0.8, 0.8)
2.2.	(0.185, 0.262, 0.448, 0.587; 1, 1) (0.2, 0.278, 0.426, 0.554; 0.8, 0.8)	(0.041, 0.05, 0.077, 0.106; 1, 1) (0.04, 0.05, 0.07, 0.098; 0.8, 0.8)	(0.044, 0.055, 0.084, 0.111; 1, 1) (0.046, 0.057, 0.08, 0.1; 0.8, 0.8)	(0.038, 0.043, 0.061, 0.076; 1, 1) (0.04, 0.05, 0.06, 0.07; 0.8, 0.8)
2.3.	(0.033, 0.046, 0.091, 0.137; 1, 1) (0.035, 0.05, 0.08, 0.124; 0.8, 0.8)	(0.034, 0.046, 0.083, 0.122; 1, 1) (0.036, 0.049, 0.08, 0.1; 0.8, 0.8)	(0.019, 0.025, 0.046, 0.07; 1, 1) (0.02, 0.026, 0.04, 0.06; 0.8, 0.8)	(0.065, 0.1, 0.22, 0.366; 1, 1) (0.072, 0.108, 0.2, 0.323; 0.8, 0.8)
3.1.	(0.029, 0.042, 0.086, 0.13; 1, 1) (0.03, 0.045, 0.08, 0.116; 0.8, 0.8)	(0.102, 0.161, 0.324, 0.462; 1, 1) (0.113, 0.174, 0.3, 0.425; 0.8, 0.8)	(0.038, 0.052, 0.095, 0.138; 1, 1) (0.041, 0.055, 0.09, 0.126; 0.8, 0.8)	(0.134, 0.215, 0.483, 0.71; 1, 1) (0.153, 0.24, 0.45, 0.649; 0.8, 0.8)
3.2.	(0.015, 0.02, 0.035, 0.053; 1, 1) (0.016, 0.021, 0.03, 0.048; 0.8, 0.8)	(0.068, 0.102, 0.198, 0.295; 1, 1) (0.075, 0.109, 0.2, 0.27; 0.8, 0.8)	(0.015, 0.02, 0.035, 0.053; 1, 1) (0.02, 0.02, 0.03, 0.05; 0.8, 0.8)	(0.163, 0.269, 0.543, 0.78; 1, 1) (0.184, 0.3, 0.51, 0.72; 0.8, 0.8)
3.3.	(0.05, 0.059, 0.09, 0.115; 1, 1) (0.052, 0.061, 0.09, 0.108; 0.8, 0.8)	(0.061, 0.089, 0.23, 0.325; 1, 1) (0.07, 0.099, 0.2, 0.294; 0.8, 0.8)	(0.038, 0.047, 0.081, 0.115; 1, 1) (0.04, 0.05, 0.076, 0.105; 0.8, 0.8)	(0.184, 0.262, 0.492, 0.632; 1, 1) (0.2, 0.28, 0.46, 0.59; 0.8, 0.8)
Total fuzzy weight	(0.018, 0.052, 0.372, 1.069; 1, 1) (0.023, 0.064, 0.3, 0.826; 0.8, 0.8)	(0.026, 0.074, 0.496, 1.336; 1, 1) (0.034, 0.091, 0.4, 1.044; 0.8, 0.8)	(0.023, 0.061, 0.384, 1.062; 1, 1) (0.03, 0.074, 0.32, 0.83; 0.8, 0.8)	(0.035, 0.09, 0.552, 1.46; 1, 1) (0.045, 0.11, 0.454, 1.14; 0.8, 0.8)
Total defuz. weight	0.331	0.426	0.337	0.471

Total norm. defuz. weight	14.868%	19.119%	15.124%	21.128%

Table 15. Local and overall priorities of alternatives E, F, G

	E	F	G
1.1.	(0.025, 0.042, 0.123, 0.257; 1, 1) (0.029, 0.046, 0.109, 0.212; 0.8, 0.8)	(0.019, 0.03, 0.082, 0.161; 1, 1) (0.021, 0.033, 0.073, 0.134; 0.8, 0.8)	(0.013, 0.018, 0.04, 0.077; 1, 1) (0.014, 0.019, 0.036, 0.065; 0.8, 0.8)
1.2.	(0.029, 0.046, 0.106, 0.192; 1, 1) (0.033, 0.05, 0.097, 0.165; 0.8, 0.8)	(0.021, 0.028, 0.055, 0.091; 1, 1) (0.022, 0.03, 0.05, 0.08; 0.8, 0.8)	(0.024, 0.033, 0.076, 0.131; 1, 1) (0.026, 0.036, 0.068, 0.111; 0.8, 0.8)
1.3.	(0.283, 0.378, 0.56, 0.674; 1, 1) (0.302, 0.4, 0.532, 0.648; 0.8, 0.8)	(0.032, 0.037, 0.051, 0.067; 1, 1) (0.033, 0.038, 0.049, 0.063; 0.8, 0.8)	(0.032, 0.037, 0.051, 0.067; 1, 1) (0.033, 0.038, 0.049, 0.063; 0.8, 0.8)
2.1.	(0.015, 0.02, 0.037, 0.06; 1, 1) (0.016, 0.021, 0.034, 0.053; 0.8, 0.8)	(0.128, 0.193, 0.392, 0.613; 1, 1) (0.14, 0.208, 0.363, 0.553; 0.8, 0.8)	(0.177, 0.282, 0.546, 0.772; 1, 1) (0.198, 0.304, 0.509, 0.715; 0.8, 0.8)
2.2.	(0.038, 0.043, 0.061, 0.076; 1, 1) (0.039, 0.045, 0.058, 0.072; 0.8, 0.8)	(0.226, 0.292, 0.461, 0.589; 1, 1) (0.239, 0.306, 0.44, 0.559; 0.8, 0.8)	(0.042, 0.049, 0.068, 0.086; 1, 1) (0.044, 0.05, 0.066, 0.081; 0.8, 0.8)
2.3.	(0.014, 0.017, 0.031, 0.049; 1, 1) (0.014, 0.018, 0.029, 0.044; 0.8, 0.8)	(0.135, 0.2, 0.409, 0.6; 1, 1) (0.148, 0.216, 0.377, 0.543; 0.8, 0.8)	(0.179, 0.272, 0.537, 0.747; 1, 1) (0.2, 0.295, 0.5, 0.693; 0.8, 0.8)
3.1.	(0.064, 0.094, 0.194, 0.316; 1, 1) (0.07, 0.1, 0.179, 0.28; 0.8, 0.8)	(0.046, 0.064, 0.128, 0.209; 1, 1) (0.049, 0.068, 0.118, 0.184; 0.8, 0.8)	(0.05, 0.067, 0.13, 0.197; 1, 1) (0.053, 0.071, 0.12, 0.175; 0.8, 0.8)
3.2.	(0.088, 0.146, 0.327, 0.538; 1, 1) (0.099, 0.159, 0.3, 0.48; 0.8, 0.8)	(0.047, 0.067, 0.145, 0.263; 1, 1) (0.051, 0.072, 0.132, 0.227; 0.8, 0.8)	(0.056, 0.077, 0.145, 0.225; 1, 1) (0.06, 0.082, 0.135, 0.203; 0.8, 0.8)
3.3.	(0.111, 0.129, 0.195, 0.233; 1, 1) (0.117, 0.135, 0.186, 0.222; 0.8, 0.8)	(0.059, 0.074, 0.131, 0.175; 1, 1) (0.063, 0.078, 0.122, 0.159; 0.8, 0.8)	(0.064, 0.078, 0.137, 0.167; 1, 1) (0.067, 0.082, 0.127, 0.154; 0.8, 0.8)
Total weight	(0.021, 0.049, 0.283, 0.789; 1, 1) (0.026, 0.059, 0.233, 0.604; 0.8, 0.8)	(0.015, 0.035, 0.227, 0.681; 1, 1) (0.018, 0.042, 0.183, 0.509; 0.8, 0.8)	(0.016, 0.037, 0.232, 0.652; 1, 1) (0.019, 0.044, 0.188, 0.496; 0.8, 0.8)
Total defuzzy weight	0.251	0.208	0.205
Total norm. defuzzy weight	11.347%	9.334%	9.079%

8. Discussion on Fuzzy-Type-2 AHP Results

Now, we see that assistant D has higher priority weigth than B and difference between them (2%) is suitable for experts. In addition, it can be noticed that E should be in the top five group, for sure (difference is also about 2%). Thus, Fuzzy AHP does not change ranks of alternatives but makes it clearer. It means that more reliable results are maintained since interval type-2 fuzzy sets can better represent uncertainties.

It is important to note that, contrary to the common belief, the system does not determine the decision we should make, rather, the results should be interpreted as a blueprint of preference and alternatives based on the level of importance obtained

for the different criteria taking into consideration our comparative judgments. In other words, the AHP methodology allows us to determine which alternative is the most consistent with our criteria and the level of importance that we give them.

Taking this point into account, Sensitivity Analysis is used. It performs a "what-if" analysis to see how the final results would have changed if the weights of the criteria would have been different **Указан недопустимый источник.**.

Let's start with a goal of finding the best teaching assistant. The first criterion has the highest weight in our results ($\approx 50\%$). If we decrease its weight and proportionally increase other weights then D will still be a leader. In this case D will have even more clear-cut victory. Otherwise, if we increase weight of this criterion up to 60% and more, then B will become a new leader. However, stakeholders come to one opinion that no one criterion should cost more than a half and they has highlighted that the first criterion (professional skills) should stay as the most important one.

It means that weight of the first criterion should be in the next approximate range [33%; 50%].

Let's now tune proportions of the second and the third criteria. Calculations show that D can stop be a winner only and only if the third criterion will cost more than the second. Thus, this point was brought to expert discussion and they have unanimously decided that personal qualities (third criterion) should be appreciated higher than communicative ones.

Another important note is change of proportions of subcriteria inside their criteria. There are no strong disputes about subcriteria weights, experts' opinions differ no more than 10%. In this case change of subcriteria preferences in that range does not influence on the leader.

It means that there is no opportunity to have another leader than D by introducing small changes in current proportions of criteria weights.

At the same time, there is a complex situation with choosing top five assistants group. Analysis shows that four assistants are determined clearly. They are A, B, C, and D. The fifth assistant can be either E or G.

Calculations reveal that position of assistant G is directly connected with the second criteria and if its weight is equal or more than 15% than G will be in top five group instead of E. However, now second criterion has only nearly 10%.

Finally, after Sensitivity Analysis is done, next recommendations for the experts are given:

- to choose assistant D as a winner;
- to prolongate contracts with A, B, C and D;
- to prolongate contract with E if experts think that personal qualities should be at least twice more important than communicating skills (finally, communicating skills should have a weight less than 15);
- to prolongate contract with G, in other case.

9. Final Result and Conclusion

Taking into consideration recommendations mentioned above, group of teachers has decided to follow first two instructions. They have selected D as the best teaching assistant on the course of «Discrete mathematics». Also, they have prolongated contracts with D and A, B, and C assistants.

The main important step now is to choose the fifth assistant. Before making a choice, experts decide to use a retrospective and to look through all methods that were applied earlier. Lecture of the course noticed that since A, B, C, and D assistants are already confirmed it means that nobody will be responsible for communication with students (answering questions, having consultations) because assistant F did it before. However, now there is a choice between either E or G. And in this case G demonstrates a clear superiority compared with others as he/she is one of the top in this kind of work. Finally, G is chosen.

At the very beginning teachers wanted to choose assistant A as the best teaching assistant. However, the soft methods of analysis helped us to choose another assistant. Also, neither AHP nor Fuzzy AHP chose G teaching assistant as the 5th best assistant in the group. Only a sound logic helped us to do this.

The application of methods of system analysis can help to make a decision but it does not make a choice for us. We should look carefully at the results of system analysis methods, but make a balanced and considered decision.

References

[1] J. Lopa. Using Undergraduate Students as Teaching Assistants. The Professional & Organizational Development Network in Higher Education, vol. 21, 2009, pp. 50-62.

[2] M. Asad, S. Kermani and H. Hora. A Proposed Framework for Evaluating Student's Performance and Selecting the Top Students in E-Learning System, Using Fuzzy AHP Method. In Proceedings of the International Conference on Management, Economics and Humanities, Istanbul-Turkey, 2015.

[3] T. Saaty. There is no mathematical validity for using fuzzy number crunching in the analytic hierarchy process. Journal of Systems Science and Systems Engineering, vol. 15, no. 4, 2006, pp. 457-464.

[4] Monk and S. Howard. The Rich Picture: A Tool for Reasoning about Work Context. [Online]. Available: http://www-users.york.ac.uk/~am1/RichPicture.pdf, accessed 12.05.2018.

[5] Changing Minds. CATWOE. [Online]. Available: http://creatingminds.org/tools/catwoe, accessed 12.05.2018.

[6] M. Velasquez and P. Hester. An Analysis of Multi-Criteria Decision Making Methods. International Journal of Operations Research, vol. 10, no. 2, 2013, pp. 56-66.

[7] T. Saaty. The Analytic Hierarchy Process, New York: McGraw Hill, 1980.

[8] C. Boender and J. De Graan. Multicriteria Decision Analysis with Fuzzy Pairwise Comparisons. Fuzzy Sets and Systems, vol. 29, 1989, pp. 133-143.

[9] D. Chang. Applications of the Extent Analysis Method on Fuzzy-AHP. European Journal of Operational Research, vol. 95, 1996, pp. 649-655.

[10] J. Mendel and R. John. Type-2 fuzzy sets made simple. IEEE Transactions on Fuzzy Systems, vol. 10, no. 2, 2002. pp. 117-127.

[11] C. Kahraman and B. Öztayşi. Fuzzy analytic hierarchy process with interval type-2 fuzzy sets. Knowledge-Based Systems, vol. 59, 2014, pp. 48-57.

[12] E. Mu and M. Pereyra-Rojas. An Introduction to the Analytic Hierarchy Process (AHP). Using Super Decisions, vol. 2, New York: Springer, 2016.

Применение методов системного анализа к оцениванию работы учебных ассистентов

Е.Н. Береснева <eberesneva@hse.ru>
М.К. Горденко <mgordenko@hse.ru>
Департамент программной инженерии,
Национальный исследовательский университет "Высшая школа экономики",
101000, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. В этой статье представлены результаты применения различных методов системного анализа (CATWOE, Rich Picture, AHP, Fuzzy AHP) для оценки учебных ассистентов для преподавателей. В статье рассмотрено применение soft- и hard-методов системного анализа. Методы системного анализа рассматриваются на примере реализации программы «Учебный ассистент» в Национальном Исследовательском Университете «Высшая школа экономики» (НИУ ВШЭ) на дисциплине «Дискретная математика». В статье показан процесс взаимодействия преподавателей с учениками и преподавателями в форме Rich Picture. Определены связующие мероприятия, встречи и даже отчеты, которые предоставляют ассистенты преподавателю. Затем описано каким образом были выбраны критерии для оценки ассистентов и оценена важность каждого критерия. Были определены три группы критериев: профессиональные навыки, навыки общения, личные качества. Каждая группа имеет некоторые подкритерий, которые были определены посредством уточняющих встреч и мозгового штурма. Также в работе был определен собственный метод оценки, который явился пререквизитом для AHP и позволивший сразу же отбросить наиболее неперспективных ассистентов. Кроме того, рассматривается применение методов AHP и Fuzzy AHP типа 2 для оценки учебных ассистентов. Выявлены сильные и слабые стороны каждого метода. Также показано, что, несмотря на мощь методов системного анализа, необходимо использовать здравый смысл и логику. Нельзя полагаться только на числа, полученные методами системного анализа, необходимо затем производить анализ результатов. В процессе работы выбирается лучший учебный ассистент, и определяется группа лучших учебных ассистентов.

Ключевые слова: системный анализ; комбинация soft и hard методов; многокритериальное принятие решений; AHP; Fuzzy Type-2 AHP; нечеткие множества.

DOI: 10.15514/ISPRAS-2018-30(3)-18

Для цитирования: Береснева Е.Н., Горденко М.К. Применение методов системного анализа к оцениванию работы учебных ассистентов. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 251-270 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-18

Список литературы

[1] J. Lopa. Using Undergraduate Students as Teaching Assistants. The Professional & Organizational Development Network in Higher Education, vol. 21, 2009, pp. 50-62.

[2] M. Asad, S. Kermani and H. Hora. A Proposed Framework for Evaluating Student's Performance and Selecting the Top Students in E-Learning System, Using Fuzzy AHP Method. In Proceedings of the International Conference on Management, Economics and Humanities, Istanbul-Turkey, 2015.

[3] T. Saaty. There is no mathematical validity for using fuzzy number crunching in the analytic hierarchy process. Journal of Systems Science and Systems Engineering, vol. 15, no. 4, 2006, pp. 457-464.

[4] Monk and S. Howard. The Rich Picture: A Tool for Reasoning about Work Context. [Online]. Available: http://www-users.york.ac.uk/~am1/RichPicture.pdf, accessed 12.05.2018.

[5] Changing Minds. CATWOE. [Online]. Available: http://creatingminds.org/tools/catwoe, accessed 12.05.2018.

[6] M. Velasquez and P. Hester. An Analysis of Multi-Criteria Decision Making Methods. International Journal of Operations Research, vol. 10, no. 2, 2013, pp. 56-66.

[7] T. Saaty. The Analytic Hierarchy Process, New York: McGraw Hill, 1980.

[8] C. Boender and J. De Graan. Multicriteria Decision Analysis with Fuzzy Pairwise Comparisons. Fuzzy Sets and Systems, vol. 29, 1989, pp. 133-143.

[9] D. Chang. Applications of the Extent Analysis Method on Fuzzy-AHP. European Journal of Operational Research, vol. 95, 1996, pp. 649-655.

[10] J. Mendel and R. John. Type-2 fuzzy sets made simple. IEEE Transactions on Fuzzy Systems, vol. 10, no. 2, 2002. pp. 117-127.

[11] C. Kahraman and B. Öztayşi. Fuzzy analytic hierarchy process with interval type-2 fuzzy sets. Knowledge-Based Systems, vol. 59, 2014, pp. 48-57.

[12] E. Mu and M. Pereyra-Rojas. An Introduction to the Analytic Hierarchy Process (AHP). Using Super Decisions, vol. 2, New York: Springer, 2016.

Static dependency analysis for semantic data validation

D.V. Ilyin <denis.ilyin@ispras.ru>
N.Yu. Fokina <nfokina@ispras.ru>
V.A. Semenov <sem@ispras.ru>
Ivannikov Institute for Systems Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia

Abstract. Modern information systems manipulate data models containing millions of items, and the tendency is to make these models even more complex. One of the most crucial aspects of modern concurrent engineering environments is their reliability. The principles of ACID (atomicity, consistency, isolation, durability) are aimed at providing it, but directly following them leads to serious performance drawbacks on large-scale models, since it is necessary to control the correctness of every performed transaction. In the paper, a method for incremental validation of object-oriented data is presented. Assuming that a submitted transaction is applied to originally consistent data, it is guaranteed that the final data representation is also consistent if only the spot rules are satisfied. To identify data items subject to spot rule validation, a bipartite data-rule dependency graph is formed. To automatically build the dependency graph a static analysis of the model specifications is proposed to apply. In the case of complex object-oriented models defining hundreds and thousands of data types and semantic rules, the static analysis seems to be the only way to realize the incremental validation and to make possible to manage the data in accordance with the ACID principles.

Keywords: information systems; ACID; data consistency management; EXPRESS

DOI: 10.15514/ISPRAS-2018-30(3)-19

For citation: Ilyin D.V., Fokina N.Yu., Semenov V.A. Static dependency analysis for semantic data validation. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 271-284. DOI: 10.15514/ISPRAS-2018-30(3)-19

1. Introduction

Management of semantically complex data is one of the challenging problems tightly connected with emerging information systems such as concurrent engineering environments and product data management systems [1-4]. Although transactional guarantees ACID (Atomicity, Consistency, Isolation, and Durability) are widely recognized and recommended for any information system, it is difficult to maintain the consistency and integrity of data driven by complex object-oriented

models. Often such models are specified in EXPRESS language being part of the STEP standard on industrial automation systems and integration (ISO 10303). To be unambiguously interpretable by different systems the data must satisfy numerous semantic rules imposed by formal models. Maintaining data consistency and ensuring system interoperability become a serious computational problem. Full semantic validation requires extremely high costs, often exceeding the processing time of individual transactions. Periodic validation is possible, but at a high risk of violating rules and losing actual data.

The paper presents an effective method for incremental validation of object-oriented data. An idea of incremental checks is well-understood and was successfully implemented for the validation of such specific data as UML charts, XML documents, deductive databases [5-7]. Unlike the aforementioned results, the proposed method can be applied to semantically complex data driven by arbitrary object-oriented models.

Assuming that a submitted transaction is applied to originally consistent data, it is guaranteed that the final data representation is also consistent if only the spot rules are satisfied. To identify data items subject to spot rule validation, a bipartite data-rule dependency graph is formed. To automatically build the dependency graph a static analysis of the model specifications is proposed to apply. In the case of large-scale models defining hundreds and thousands of data types and semantic rules, static analysis seems to be the only way to realize the incremental validation and to make possible to effectively manage the data in accordance with the ACID principles.

The structure of the paper is as follows. In section 2, we will shortly overview EXPRESS language with an emphasis on the data types and the rule categories admitted by the language. Formal definitions of model-driven data, rules and transactions are also provided. In section 3, we will present a complete validation routine and then explain how an incremental validation can be arranged using the proposed dependency graph. This is accompanied by an example of the model specification. In conclusion, we summarise benefits of the proposed validation method and outdraw future efforts.

2. Product data and transactions

2.1 EXPRESS language

Product data models and, particularly, semantic rules can be specified formally in EXPRESS (ISO 2004) language [8]. This object-oriented modeling language provides a wide range of declarative and imperative constructs to define both data types and constraints imposed upon them. The supported data types can be subdivided into the following groups: simple types (character, string, integer, float, double, Boolean, logical, binary), aggregate types (set, multi-set, sequence, array), selects, enumerations, and entity types.

Depending on the definition context, three basic sorts of constraints are distinguished in the modeling language: rules for simple user-defined data types, local rules for object types, and global rules for object type extents. Depending on the evaluation context these imply the following semantic checks:

- attribute type compliance (R_0);
- limited widths of strings and binaries (R_1, R_2);
- size of aggregates (R_3);
- multiplicity of direct and inverse associations in objects (R_4, R_5);
- uniqueness of elements in sets, unique lists and arrays (R_6);
- mandatory attributes in objects (R_7);
- mandatory elements in aggregates excluding sparse arrays (R_8);
- value domains for primitive data types (R_9);
- value domains restricting and interrelating the states of separate attributes within objects (R_{10} or so-called local rules);
- uniqueness of attribute values (optionally, their groups) on object type extents (R_{11} or uniqueness rules);
- value domains restricting and interrelating the states of whole object populations (R_{12} or so-called global rules). Value domains can be specified in a general algebraic form by means of all the variety of imperative constructs available in the language (control statements, functions, procedures, etc.).

Certainly, each product model defines own data types and rules. Therefore, semantic validation methods and tools should be developed in a model-driven paradigm allowing their application for any data whose model is formally specified in EXPRESS language. For a more detailed description refer to the mentioned above standard family which regulates the language.

2.2 Formalization of models, data and transactions

An object-oriented data model M can be formally considered as a triple $M = \langle T \cup \prec \cup R \rangle$, where the types $T = \{C \cup S \cup A \cup \dots\}$ are classes C, simple types S, aggregates A and other constructed structures allowed by EXPRESS. Generalization/specialization relations $\prec$ are defined among these types. Each class $c \in C$ defines a set of attributes in the form $c.a{:}\, C \mapsto T$. The attributes $c.a{:}\, C \mapsto C$, $c.a{:}\, C \mapsto aggregate(C)$ are single and multiple associations which play role of object references. The rules $R = \{R_0 \cup R_1 \cup R_2 \cup \dots \cup R_{12}\}$ define the value domains of typed data in an algebraic way in accordance with EXPRESS. The rules are subdivided into 12 categories enumerated above. Let us define the key concepts that are used in further consideration.

An object-oriented dataset $x = \{o_1, o_2, \dots\}$ is said to be driven by the model $M\langle T, \prec, R\rangle$ if all the objects belong to its classes: $\forall\, o \in x \to typeof(o) \in C \subset T$.

Let a dataset x is driven by the model $M\langle T, \prec, R\rangle$. All the objects $\{o^*\} \subset x$ such that $subtypeof(o^*) = c \in C \subset T$ are called an extent of the class c on the dataset x. A query returning the class extent c on the dataset x is called the extent query and is designated as $Q_{extent}(x, c)$.

Let a dataset x is driven by the model $M\langle T, \prec, R\rangle$. An object set $\{o^*\} \subset x$, $typeof(o^*) = c^* \in C \subset T$ is said to be interlinked with the objects $\{o\} \subset x$, $typeof(o) = c \in C \subset T$ along the association $c.a$ if $\forall o \in \{o\}, o.a \subset \{o^*\}$, $\forall o^* \in \{o^*\} \to \exists o \in \{o\}: o^* \in o.a$. We will denote that as $\{o\} \xrightarrow{c.a} \{o^*\}$.

Let a dataset x is driven by the model $M\langle T, \prec, R\rangle$. An object set $\{o^*\} \subset x$, $typeof(o^*) = c^* \in C \subset T$ is said to be interlinked with the objects $\{o\} \subset x$, $typeof(o) = c \in C \subset T$ along the route $\{c.a\}$ if $\exists\, \{o'\} \subset x, \{o''\} \subset x$, …, so that $\{o\} \xrightarrow{c.a} \{o'\} \xrightarrow{c'.a'} \{o''\} \xrightarrow{c''.a''} \dots \to \{o^*\}$. A query returning the objects $\{o^*\}$ interlinked with a given set $\{o\}$ along the route $\{c.a\}$ is called the route query and is designated as $Q_{route}(x, \{o\}, \{c.a\})$. A query returning the objects $\{o\}$ by a given object set $\{o^*\}$ is called the reverse route query and is designated as $Q_{route}(x, \{o^*\}, rev\ \{c.a\})$.

The object set $x = \{o_1, o_2, \dots\}$ driven by the model $M\langle T, \prec, R\rangle$ is called consistent if all the rules being instantiated and evaluated are satisfied on this data set: $\forall\, r \in R \to r(x) = true$.

Finally, let us introduce the concept of the delta as a specific representation of transactions. Each delta $\Delta(x', x)$ aggregates the changes happened in the dataset $x' = \{o'_1, o'_2, \dots\}$ compared with its original revision $x = \{o_1, o_2, \dots\}$. It is assumed that both revisions are driven by the same model and the objects have unique identifiers that allows to uniquely map the objects and to compute delta in a formal way as $\Delta(x', x) = delta(x', x)$. The delta can be arranged as bidirectional one and then any of the revisions can be restored by the given other: $x' = apply(x, \Delta)$ and $x = apply(x', \Delta^{-1})$.

The delta is represented as a set of elementary and compound changes $\Delta = \{\delta\}$, where each change can be either the creation of an object, or its deletion or modification designated as $\delta_{new(o)}$, $\delta_{del(o)}$, $\delta_{mod(o)}$ correspondingly. The modification, in turn, is represented as a change in the attributes $\delta_{mod(o)} = \{\delta_{mod(o.a)}\}$ that in the case of aggregates is represented by the operations of insertion, removal and modification of the elements $\delta_{mod(o.a)} = \{\delta_{ins(o.a[])}, \delta_{rem(o.a[])}, \delta_{mod(o.a[])}\}$. In what follows, we assume that each creation operation in the delta representation is complemented by the operations of initializing the attributes that are equivalent to the modification operations. Each deletion operation is supplemented by the operations of resetting the attributes to an undefined state, also representable by the modification operations. Regardless of the way, the delta is structured, only elementary operations are taken into account in the context of the studied validation problems.

3. Validation

3.1 Complete validation

The complete validation routine is provided below (see Figure 1). In a cycle on all objects their attributes are checked against the rules of the categories $R_1 \cup R_2 \cup ... \cup R_9$. The checks are performed individually for each attribute provided that the corresponding rules are imposed on their types. In case of detected violations, the error messages are logged. Rules R_{10} are evaluated for entire objects in the same loop. The second cycle is formed due to the need for checks of uniqueness rules R_{11}. Since these rules are declared inside the class definitions, an additional cycle is arranged on the model classes. The rules are evaluated on the class extents. Finally, the third cycle allows to check global rules R_{12} which are defined directly in the model. Such checks are performed for the corresponding class extents.

```
for each object o ∈ x in dataset
    for each attribute o.a in object
        for each attribute rule ∈R0 U R1 U R2 U … U R9 defined for typeof( o.a )
            check rule(o.a), log if violated
    for each local rule ∈ R10 defined for typeof( o )
        check rule( o ), log if violated
for each class c ∈ C defined in model
    for each uniqueness rule ∈ R11 defined for class c
        check rule( Q_extent( x, rule.c ) ), log if violated
for each global rule ∈ R12 defined in model
    check rule( Q_extent( x, rule.c1 ), Q_extent( x, rule.c2 ),… ), log if violated
```

Fig. 1. Complete validation routine

As mentioned above, complete validation of semantically complex product data is a computationally costly task that can cause performance degradation when processing transactions. Incremental validation makes it possible to reduce the amount of checks to be performed.

3.2 Incremental validation

The proposed incremental validation method is based on the idea of localizing spot rules that can be affected by a transaction and generating a set of semantic checks that is sufficient to detect all potential violations. For this purpose, the dependency graph is built by a given specification of the data model in EXPRESS language. For brevity, we just explain that this structure represents and omit the details of how it can be formed using static analysis of the specification.

The dependency graph is a bipartite graph whose nodes represent the kinds of transaction operations and the categories of semantic rules both defined by the underlying model. An operation node is connected with the rule nodes by directed edges if only such operations can violate the rules being instantiated for particular data. Usually, the semantics of the operations imply what are the data it is applied to. Sometimes the inspected data are apriori unknown and have to be determined by

executing corresponding route queries. Therefore, each edge is formed by the dependency structure σ containing both a rule reference $\sigma.rule$ and an optional query route $\sigma.route$. In some sense, the graph reflects the transaction structure as if it contains all possible kinds of changes and the data organisation as if all data types are present and all rules are potentially suffered to violations. As mentioned above, only elementary operations are involved in the dependency analysis.

Thus, the dependency graph enables to determine spot rules that could be violated for particular data due to the accepted transaction. For example, if the node operation is a modification of the object attribute $c.a$ and a rule $r \in R_0 \cup R_1 \cup R_2 \cup ... \cup R_9$ is defined for its type, then the node $\delta_{mod(c.a)}$ is connected with the rule node r by a corresponding edge. Having a specific operation of this kind $\delta_{mod(o.a)}$, $typeof(o) = c$ in the delta representation the corresponding check $r(o.a)$ can be produced using the dependency edge.

The method of the dependency graph construction is described in more detail in the next section. Still, here we will point out some of its important features.

If the same attribute $c.a$ participates in an expression of the domain rule $r \in R_{10}$ for the class c, then the operation $\delta_{mod(o.a)}$, $typeof(o) = c$ produces the check $r(o)$ for the object o.

If the attribute $c.a$ participates in the uniqueness rule $r \in R_{11}$ defined for the class c, then another dependency edge must be associated with the operation node. In this case, the corresponding check $r(Q_{extent}(x, c))$ must be performed.

There is a more difficult case when the attribute $c.a$ participates in an expression of the domain rule $r \in R_{10}$ defined for the other class c^*. The attribute $c.a$ is assumed to be accessed by traversing associated objects along the route $\{c^*.a^*\}$ from the objects $o^* \in c^*$. Then the operation $\delta_{mod(o.a)}$, $typeof(o) = c$ induces the checks $r(o^*)$ for all $o^* \in Q_{route}(x, o, rev\{c^*.a^*\})$. To identify and perform such checks the operation node must be connected with the evaluated rule node and a route $\{c^*.a^*\}$ must be prescribed to the edge. The dependency analysis of spot rules $r \in R_{12}$ is carried out in a similar way.

Finally, we note that the operations of creating and deleting objects on the assumptions made above can only violate global rules and only in those cases if the cardinalities of class extents are computed. Considering object references as specific attribute types, it is possible to localize some spot rules more exactly. Differing operations on aggregates also leads to better localization of spot rules. For brevity we omit the details how the spot rules can be localized more carefully and provide an example in the next subsection.

```
for each elementary operation δ(o),δ(o.a) ∈ delta
    { σ } = dependency_graph( kindof( δ ) )
    for each dependency σ ∈ { σ }
        switch kindof( σ.rule )
            case attribute_rule :
                check σ.rule( o.a ), log if violated
            case local_rule :
```

```
                    { o* } = Query_route( x, o, rev (σ.route) )
                    for each o* ∈ { o* }
                        checkset.put( σ.rule( o* ) )
                case uniqueness_rule :
                    checkset.put( σ )
                case global_rule :
                    checkset.put( σ )
    for each check σ, σ(o) ∈ checkset
        switch kindof( σ.rule )
            case local_rule :
                check σ.rule( o ), log if violated
            case uniqueness_rule :
                check σ.rule( Query_extent( x, σ.rule.c ) ), log if violated
            case global_rule :
    check σ.rule( Query_extent( x, σ.rule.c1 ), Query_extent( x, σ.rule.c2 ),… ),
log if violated
```

Fig. 2. Incremental validation routine

The validation routine presented in Figure 2 consists in the sequential traversing of delta operations, determining the nodes of the operation semantics, obtaining associated spot rule nodes, evaluating the rules directly or filling the checkset for the subsequent validation. The checkset is organized as an indexed set of records each of which stores references on the validated rule, query and factual data to perform the corresponding check. The use of the checkset is motivated by the fact that some operations lead to repeated checks of the same rules. Indexing of the checkset allows you to exclude repeated records and, thus, to avoid redundant computations. At the same time, the attribute rule checks are always produced once by the modification operations and, therefore, it is more expedient to execute them immediately, without overloading the checkset.

3.3 Dependency graph construction

To construct the dependency graph, an abstract syntactic tree for the model is built. According to the retrieved data, for all attribute declarations operation nodes are built. Number and types of these nodes constructed for a single attribute depend on its type. For non-aggregate attributes $c.a$ only node $\delta_{mod}(c.a)$, representing modification of the attribute, is built. For aggregate attributes $c.a[]$ three nodes are created: (1) $\delta_{ins}(c.a[])$ – insertion of a new element; (2) $\delta_{mod}(c.a[])$ – modification of an element of the aggregate; (3) $\delta_{rem}(c.a[])$ – removal of an element.

Construction of the dependency graph proceeds with generating rule nodes. We handle construction of nodes for rules R_1-R_9 and R_{10}-R_{12} differently.

For rules R_1-R_9 we take all explicit attributes and build rule nodes for each of them. The types of rule nodes depend on the type of the attribute in question. For instance, if it is a bounded string c.*S*, we generate a $R_1(c.S)$ (R_1 – limited width of strings), connected with the node corresponding to the modification of S $\delta_{mod}(c.S)$. Similarly, if an attribute is a bounded aggregate, we construct a node of type R_4 and

connect it with the insertion $\delta_{ins}(c.a[])$ and/or removal $\delta_{rem}(c.a[])$ operation nodes of the attribute, depending on the side from which the aggregate is bounded – if it is bounded above, then only with insertion node, if below – with removal, if from both sides – with both of them.

The way of construction of rule nodes for R_{10}-R_{12} is uniform. We start with locating all local rules for R_{10}, all uniqueness rules for R_{11} and all global rules for R_{12}. For each of the rules, we find all attributes used in it. If an attribute is explicit, we only connect its modification with the rule node, and also with insertion and removal, if it is an aggregate used inside a SIZEOF operation. If an attribute is derived, we take its definition and find the attributes used in it; if inverse – we proceed with analyzing the attribute it references. For derived and explicit attributes, the analysis is performed recursively, until all the explicit attributes, directly and indirectly referenced by them, are located. Then all of them are connected with the rule node corresponding to the rule in question. If the during the analysis we find a node that is a function call, we substitute its formal parameters with actual and thus locate the attributes which are used in it; the analysis of a function body with the parameters substituted is completely identical to the analysis of a rule.

An example illustrating the constructed graph is given in the next subsection.

3.4 Example of a dependency graph

Let us consider a fragment of the EXPRESS specification of a project management system. Three classes depicted in Figure 3 – *Task*, *Link* and *Calendar* – are its core entities. The meaning of Task is self-evident; *Link* represents a connection defining a relation and execution order between two tasks. The fact that between two tasks might be only a single link of one type is reflected in uniqueness rule *ur1*. A *Calendar* defines a typical working pattern: working days, working times, holidays. The calendar can be assigned to specific tasks, and one calendar can be set as a default project calendar, that means that it will be used for tasks for which no task calendar is set. Besides that, it is possible to use an *Elapsed* calendar for a task implying that work will be performed 24/7. Global rule *SingleProjectCalendar* restricts the possible number of project calendars to no more than one. Moreover, local rule *wr3* is used to check that if a task has got a task calendar, it the reference to it must be non-null. One more local rule, *wr2*, restricts the length of an *EntityName* to be between 1 and 32 characters.

```
TYPE LinkEnum = ENUMERATION OF
    (START_START, START_FINISH, FINISH_START, FINISH_FINISH);
END_TYPE;

TYPE CalendarRuleEnum = ENUMERATION OF
    (TASK, PROJECT, ELAPSED);
END_TYPE;

FUNCTION TaskIsCyclic (T1 : Task, T2 : Task) : BOOLEAN;
    IF (SIZEOF(T1.Parent) = 0) THEN RETURN(FALSE);
    ELSE
```

```
        IF ((TaskIsCyclic(T1.Parent[1], T2) = TRUE) OR (T1 = T2))
            THEN RETURN(TRUE);
        END_IF;
    END_IF;
END_FUNCTION;
RULE SingleProjectCalendar FOR (Calendar);
WHERE
    wr1: SIZEOF(QUERY(Temp <* Calendar | Temp.isProjectCalendar =
TRUE)) <= 1;
END_RULE;

TYPE EntityName = STRING;
WHERE
    wr2: (1 <= SELF) AND (SELF <= 32);
END_TYPE;

ENTITY Task;
    ID : INTEGER;
    Name : EntityName;
    TaskCalendar : Calendar;
    CalendarRule : CalendarRuleEnum;
    Children : LIST [0:?] OF Task;
DERIVE
    TaskDuration : Duration := ?;
INVERSE
    Parent : SET [0:1] OF Task FOR Children;
    DownstreamLinks : SET [0:?] OF Link FOR Predecessor;
    UpstreamLinks : SET [0:?] OF Link FOR Successor;
WHERE
    wr3 : CalendarRule <> CalendarRuleEnum.TASK OR
EXISTS(TaskCalendar);
    wr4 : (SIZEOF(Parent) = 0)  OR (TaskIsCyclic(Parent[1], SELF) =
FALSE);
UNIQUE
    ur1 : ID;
END_ENTITY;

ENTITY Link;
    ID : INTEGER;
    LinkType : LinkEnum;
    Predecessor : Task;
    Successor : Task;
UNIQUE
    ur2 : LinkType AND Predecessor.ID AND Successor.ID;
    ur3 : ID;
END_ENTITY;

ENTITY Calendar;
    ID : INTEGER;
    Name : OPTIONAL EntityName;
    IsProjectCalendar : BOOLEAN;
UNIQUE
    ur4 : ID;
END_ENTITY;
```

Fig. 3. An example of the model specification in EXPRESS language

The dependency graph for this fragment of the specification is shown in Figure 4.

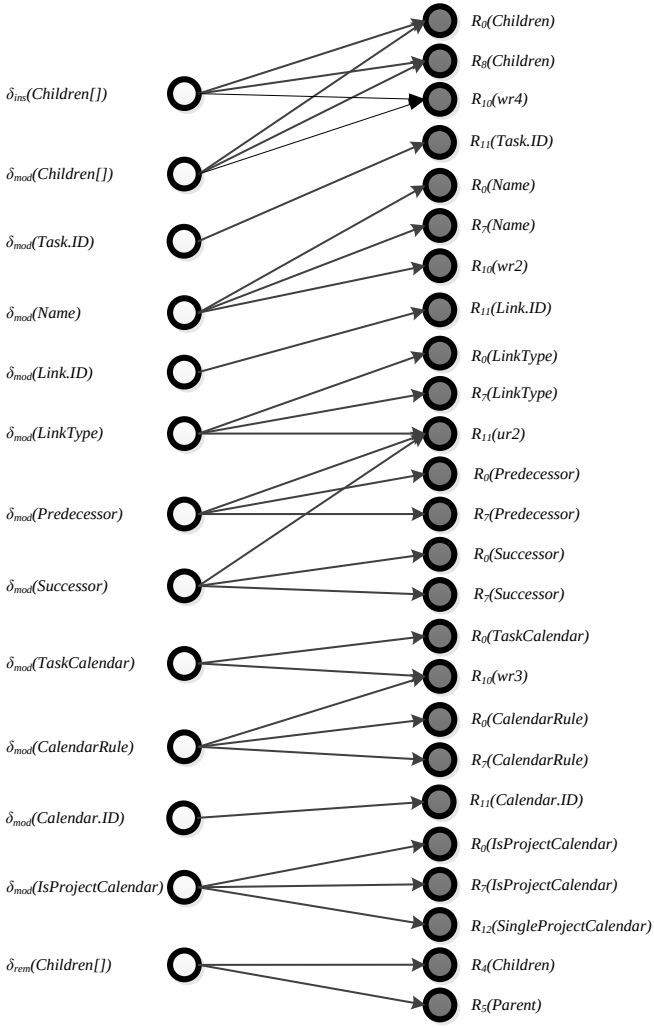


Fig. 4. A fragment of the model dependency graph

Each operation of attribute modification except for removal of elements from the list of task children is connected with the rules validating corresponding attribute type compliance R_0 and availability of defined values for mandatory attributes R_7. To avoid placement of null values to the list of mandatory elements the rule R_8 should be validated as well after the operations have been performed. The insertion cannot violate multiplicity of the direct and inverse associations as their upper borders are unlimited, but checks R_4, R_5 should be performed when an element is

removed from *Children*. Therefore, the corresponding operation nodes should be connected with the aforementioned nodes of the rules that the operations may potentially violate. As the expression for the local rule *wr3* includes the attributes *CalendarRule* and *TaskCalendar*, the nodes corresponding to the operations of modification of these attributes are connected with the *wr3* rule node. For the rule *wr2* defining the value range of the *EntityName* type, there is a connection between the *EntityName* modification node and the *wr2* rule node. The corresponding edges are assigned by the routes by traversing of which the attributes could be accessed. The expression of the global rule *SingleProjectCalendar* references only one attribute *IsProjectCalendar*, so the appropriate graph nodes are connected by the edge as well. Modification of any attribute of the *Link* class can affect its uniqueness defined by *ur2*; hence the connections between *LinkType*, *Predecessor* and *Successor* and the uniqueness rule node.

It is also possible that a change affects a constraint not directly but through an inverse association, or even a chain of them, where other classes can be involved. In this case, rules for all the chain of affected classes is added to the checkset. Furthermore, they can be affected not only by direct associations but also by the inverse. For instance, cardinality constraints on inverse aggregate attributes causes insertion of additional rule nodes to the graph.

4. Conclusion

This paper presents the incremental method of model data validation. The method is applicable for semantically complex data driven by arbitrary object-oriented models. It allows to increase the performance of semantic validation and to effectively manage the data in accordance with the ACID principles.

The planned work concerns basically the implementation of the method proposed and its evaluation for industry meaningful product data. The expected positive results will allow its wide introduction into new software engineering technologies and emerging information systems.

References

[1]. V.A. Semenov. Product Data Management with Solid Transactional Guarantees, In Transdisciplinary Engineering: A Paradigm Shift Series Advances in Transdisciplinary Engineering, IOS Press, 2017, pp. 592-599.

[2]. L. Lämmer and M. Theiss. Product Lifecycle Management, In Concurrent Engineering in the 21st Century – Foundations, Developments and Challenges, Springer, 2015, pp. 455-490.

[3]. J. Osborn. Survey of concurrent engineering environments and the application of best practices towards the development of a multiple industry, multiple domain environment. Clemson University, 2009. Accessed: 29/01/2018. Available: http://tigerprints.clemson.edu/all_theses/635/

[4]. M. Philpotts. An introduction to the concepts, benefits and terminology of product data management, Industrial Management & Data Systems, MCB University Press, vol. 96, no. 4, 1996, pp. 11–17.

[5]. X. Blanc, A. Mougenot, I. Mounier, T. Mens. Incremental Detection of Model Inconsistencies based on Model Operations. In Advanced Information Systems Engineering, CAiSE 2009, LNCS, vol. 5565, Springer, 2009, pp. 32-46.
[6]. C. Xu, C.S. Cheung, W.K. Chan. Incremental Consistency Checking for Pervasive Context. In Proc. the 28th International Conference on Software Engineering, 2006, pp. 292-301.
[7]. J. Harrison, S.W. Dietrich. Towards an Incremental Condition Evaluation Strategy for Active Deductive Databases. In Research and Practical Issues in Databases, World Scientific, 1992, pp. 81-95.
[8]. ISO 10303-11: 2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, ISO, 2004.

Статический анализ зависимостей для семантической валидации данных

Ильин Д.В. <denis.ilyin@ispras.ru>
Фокина Н.Ю. <nfokina@ispras.ru>
Семенов В.А. <sem@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Современные информационные системы манипулируют моделями данных, содержащими миллионы объектов, и тенденция такова, что эти модели постоянно усложняются. Одним из важнейших аспектов современных параллельных инженерных сред является их надежность. Принципы ACID (атомарность, согласованность, изолированность, устойчивость) направлены на ее обеспечение, однако прямое следование им приводит к серьезному снижению производительности на крупномасштабных моделях, поскольку необходимо контролировать правильность каждой выполненной транзакции. В настоящей статье представлен метод инкрементальной валидации объектно-ориентированных данных. Предполагая, что транзакция применяется к первоначально согласованным данным, гарантируется, что окончательное представление данных также будет согласованным, если только будут выполнены локальные правила. Для определения объектов данных, подлежащих проверке, формируется двудольный граф зависимостей по данным. Для автоматического построения графа зависимостей предлагается применять статический анализ спецификаций модели. В случае сложных объектно-ориентированных моделей, включающих сотни и тысячи типов данных и семантических правил, статический анализ, по-видимому, является единственным способом реализации инкрементальной валидации и обеспечения возможности управления данными в соответствии с принципами ACID.

Ключевые слова: информационные системы; ACID; управление целостностью данных; EXPRESS

DOI: 10.15514/ISPRAS-2018-30(3)-19

Для цитирования: Ильин Д.В., Фокина Н.Ю., Семенов В.А. Статический анализ зависимостей для семантической валидации данных. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 271-284 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-19

Список литературы

[1]. V.A. Semenov. Product Data Management with Solid Transactional Guarantees, In Transdisciplinary Engineering: A Paradigm Shift Series Advances in Transdisciplinary Engineering, IOS Press, 2017, pp. 592-599.

[2]. L. Lämmer and M. Theiss. Product Lifecycle Management, In Concurrent Engineering in the 21st Century – Foundations, Developments and Challenges, Springer, 2015, pp. 455-490.

[3]. J. Osborn. Survey of concurrent engineering environments and the application of best practices towards the development of a multiple industry, multiple domain environment. Clemson University, 2009. Дата обращения: 29/01/2018. Режим доступа: http://tigerprints.clemson.edu/all_theses/635/

[4]. M. Philpotts. An introduction to the concepts, benefits and terminology of product data management, Industrial Management & Data Systems, MCB University Press, vol. 96, no. 4, 1996, pp. 11–17.

[5]. X. Blanc, A. Mougenot, I. Mounier, T. Mens. Incremental Detection of Model Inconsistencies based on Model Operations. In Advanced Information Systems Engineering, CAiSE 2009, LNCS, vol. 5565, Springer, 2009, pp. 32-46.

[6]. C. Xu, C.S. Cheung, W.K. Chan. Incremental Consistency Checking for Pervasive Context. In Proc. the 28th International Conference on Software Engineering, 2006, pp. 292-301.

[7]. J. Harrison, S.W. Dietrich. Towards an Incremental Condition Evaluation Strategy for Active Deductive Databases. In Research and Practical Issues in Databases, World Scientific, 1992, pp. 81-95.

[8]. ISO 10303-11: 2004. Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual, ISO, 2004.

Simulating Behavior of Multi-Agent Systems with Acyclic Interactions of Agents[1]

[1, 2] *R.A. Nesterov <rnesterov@hse.ru, r.nesterov@campus.unimib.it>*
[1] *A.A. Mitsyuk <amitsyuk@hse.ru>*
[1] *I.A. Lomazova <ilomazova@hse.ru>*
[1] *National Research University Higher School of Economics,*
20, Myasnitskaya st., Moscow, 101000, Russia
[2] *Dipartimento di Informatica, Sistemistica e Communicazione,*
Università degli Studi di Milano-Bicocca,
Viale Sarca 336 – Edificio U14, I-20126 Milano, Italia

Abstract. In this paper, we present an approach to model and simulate models of multi-agent systems (MAS) using Petri nets. A MAS is modeled as a set of workflow nets. The agent-to-agent interactions are described by means of an interface. It is a logical formula over atomic interaction constraints specifying the order of inner agent actions. Our study considers positive and negative interaction rules. In this work, we study interfaces describing acyclic agent interactions. We propose an algorithm for simulating the MAS with respect to a given interface. The algorithm is implemented as a ProM 6 plug-in that allows one to generate a set of event logs. We suggest our approach to be used for evaluating process discovery techniques against the quality of obtained models since this research area is on the rise. The proposed approach can be used for process discovery algorithms concerning internal agent interactions of the MAS.

Keywords: Petri nets; multi-agent systems; interaction; interface; simulation; event logs.

DOI: 10.15514/ISPRAS-2018-30(3)-20

For citation: Nesterov R.A., Mitsyuk A.A., Lomazova I.A. Simulating Behavior of Multi-Agent Systems with Acyclic Interactions of Agents. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 285-302. DOI: 10.15514/ISPRAS-2018-30(3)-20

1. Introduction

Process discovery has been actively developed over recent years [1]. Many algorithms for the automatic model synthesis from event logs have been proposed [2]–[7]. They produce process models in different notations. These can be Petri nets

[1]This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16-01-00546.

[3], [6], [7], fuzzy models [2], heuristics nets [4] or BPMN models [5] and many others (see [8] for the comprehensive review of process discovery algorithms).

Discovering process models from event logs helps to use information about users and system runtime behavior for proper specification, design, and maintenance of software systems [9], [10]. This topic is increasingly attracting the attention of researchers [11]–[14]. In particular, application of process mining techniques to distributed and multi-agent software systems [15], [16] is interesting and important.

The main drawback of most algorithms is that they are not appropriate for modeling highly concurrent systems. In particular, these are multi-agent systems (MAS). Such a system consists of multiple agents executing their work independently and interacting via predefined interfaces. It makes sense to use compositional approaches to model MAS's. Fortunately, such approaches have been proposed over recent years [17], [18].

The overwhelming majority of process discovery algorithms employ different heuristics. That is why testing is used to evaluate their efficiency and validity [8]. It is performed using real-life and artificially generated event logs with suitable characteristics. The latter are prepared using *event log generators*.

In this paper, we describe a new event log generator that aims at preparing artificial event logs for MAS's. We model individual agents using workflow nets, whereas interfaces are specified using special formulae. They are constructed using a declarative formalism that we introduce to describe basic asynchronous interactions between agents. Based on agent models and a declarative interface formula our generator derives the operational semantics that describes a MAS behavior. We show that both of MAS representations are equivalent, i.e. they have the same set of possible model runs. Thus, this semantics can be used to simulate the model and generate event logs.

The *main contributions of this paper* are:

- a formalism for a declarative description of the requirements for agent interactions is defined;
- the operational semantics representing the behavior of a multi-agent system with declarative requirements for interactions of agents is defined;
- an algorithm for generating event logs from given agent models and declarative constraints on their interactions based on the operational semantics is developed;
- the approach is implemented as a prototype software and evaluated.

This paper is structured as follows. The next section gives an overview of existing approaches for generating event logs and simulating process models. Section 3 introduces main notions used in the paper. In Section 4, we describe our approach to modeling multi-agent systems with the help of Petri nets. Implementation details are discussed in Section 5, and Section 6 concludes the paper.

2. Related Work

Process Logs Generator PLG2 [19] is one of the most popular tools for generating well-structured process models represented by dependency graphs. The tool constructs models using randomly generated context-free grammars. The user should specify desired characteristics of models: a size, a number of choices, hierarchy blocks etc. The obtained model can be used to generate an event log.

Another tool that aims at randomized event log generation is *PT and Log Generator* [20]. It generates random process trees (well-structured models) containing desired number of specified workflow patterns. In particular, generated models can be constructed from sequences, AND/XOR/OR splits and joins, as well as structured loops. The algorithm can also randomly insert elements representing activities. The tool also generates the desired number of logs from automatically constructed models.

The problem of the randomized process model generation has also been addressed by Yan, Dijkman, and Grefen in [21]. However, they have not considered event log generation within the context of their approach.

The main goal of the tools discussed above is the randomized testing using sets of models and event logs. However, in some cases there is a need to generate event logs from specific process models that have been prepared on the basis of the real data or expert knowledge. If this is the case, one can use the tool *GENA* [22]. It aims at generating sets of event logs from a Petri net model. The approach allows users to use preferences to influence a control-flow and to artificially introduce a randomized noise into an event log. The improved version of GENA can generate event logs from BPMN 2.0 models [23]. Most basic BPMN constructs are supported: tasks, gateways, messages, pools, lanes, data objects.

Colored Petri nets can be used to generate event logs [24]. Authors have developed the extension for *CPN Tools* that can generate randomized event logs based on a given colored Petri net. The main drawback of this approach is that it implies writing Standard ML scripts, which leads to possible problems during tool adaptation for a specific task. Moreover, this approach and GENA do not support multi-agent systems with independent asynchronous agents.

Declarative process models might also be used to generate event logs [25]. This approach is based on construction of a finite automaton using a *Declare* process model. The tool can generate a specified number of strings accepted by this automaton. Strings are generated using the automaton and its randomized execution. Afterwards, each string is transformed into a log trace with necessary attributes. This tool is useful, when the only information about the process is the set of constraints. This approach is also not appropriate for the MAS simulation as we suggest, because it does not support the imperative control-flow description of individual agents.

In this paper, we propose an extension to the GENA tool that is supposed to be used for generating event logs by simulating MAS models, because the tools described above cannot fully support this feature.

3. Preliminaries

Let $\mathbb{N}$ denote the set of all non-negative integers, A^+ – the set of all finite non-empty sequences over a set A, and $A^* = A^+ \cup \{\varepsilon\}$, where ε is the empty sequence. For a subset $B \subset A$, the projection of $\sigma \in A^*$ on a set B, denoted $\sigma|_B$, is the subsequence of σ including all elements belonging to B.

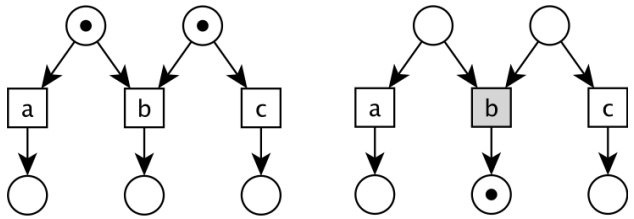
3.1 Petri Nets

A *Petri net* is a triple $N = (P, T, F)$, where P and T are two disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation. Pictorially, places are shown by circles, transitions – by boxes, whereas the flow relation is depicted using directed arcs (see Fig. 1 for an example).

We suppose that transitions of a Petri net are labeled with activity names from $\mathcal{A} \cup \{\tau\}$, where $\mathcal{A}$ is a set of visible activity names, and τ is a label for an invisible action. Labels are assigned to transitions via a labeling function $\lambda: T \to \mathcal{A} \cup \{\tau\}$.

A marking (*state*) of a Petri net N is a function $m: P \to \mathbb{N}$ assigning numbers to places. A marking m is designated by putting $m(p)$ black dots into each place p. By m_0 we denote the initial marking.

Let $X = P \cup T$. For $x \in X$, ${}^\bullet x = \{y \in X \mid (y, x) \in F\}$ is the set of *input* nodes of x in N, and $x^\bullet = \{y \in X \mid (x, y) \in F\}$ is the set of its *output* nodes.



(a) initial marking (b) transition *b* fires

Fig. 1. A Petri net

A marking m *enables* a transition $t \in T$ iff there is at least one token in all places which are input for t. An enabled transition may *fire* yielding a new marking m' (denoted $m[t\rangle m'$), consuming one token from each of its input places and producing a token into each of its output places (see Fig. 1b).

A sequence $w = t_1 t_2 ... t_n$ over T is a *firing sequence* iff $m_0[t_1\rangle m_1[t_2\rangle \ldots m_{n-1}[t_n\rangle m_n$ (denoted $m_0[w\rangle m_n$).

Let $w = t_1 t_2 ... t_n$ be a firing sequence of the net N, λ – a labeling function over a set of activity names $\mathcal{A}$. Define $\lambda(w) = \lambda(t_1)\lambda(t_2)\ldots\lambda(t_n)$. Then $\lambda(w)|_{\mathcal{A}}$ is called an (observable) *run* in N.

A marking m is *reachable* iff $\exists\, w \in T^*: m_0[w\rangle m$. A reachable marking is called *dead* if it does not enable any transition.

Workflow nets (WF-nets) form a subclass of Petri nets used for business process modeling. A *Petri net* is a triple $N = (P, T, F, m_0)$ is a WF-net iff:

- there is a single source place i and a single source place f, s.t. $\bullet i = f^\bullet = \varnothing$;
- each node in $P \cup T$ lies on a path from i to f.

The initial marking m_0 of a WF-net contains exactly one token in its source place i.

3.2 Event Logs

A *multiset* over a set A is a map $B: A \to \mathbb{N}$. The set of all multisets over A is denoted by $\mathcal{B}(A)$.

Let $\mathcal{A}$ be a set of activity names. A *trace* σ over $\mathcal{A}$ is defined as a finite non-empty sequence over $\mathcal{A}$, i.e. $\sigma \in \mathcal{A}^+$. An *event log* L over $\mathcal{A}$ is a finite multiset of traces, i.e. $L \in \mathcal{B}(\mathcal{A}^+)$.

4. Modeling Multi-Agent Systems

In this section, we present formalism for modeling multi-agent systems consisting of several asynchronously interacting agents.

A model for a system of k agents will consist of k WF-nets $N_1, N_2, \ldots, N_k$, representing behavior of individual agents (called *agent nets*), and constraints on their asynchronous interaction $\mathcal{I}$ (called *interface*).

We assume that transitions of agent nets have individual labels. In other words, different agents implement different activities. We also assume that agent interactions are *acyclic*, namely, activities in interaction constraints do not belong to cycles and therefore occur in each system run not more than once.

Interfaces are defined as positive logical formulae over atomic constraints. Let us give the exact definitions.

Let $N_1, N_2, \ldots, N_k$ be agent nets with pairwise disjoint sets of activity names $\lambda_1(T_1)$, $\lambda_2(T_2), \ldots, \lambda_k(T_k)$ respectively. We define two types of *atomic constraints*, namely $A \triangleleft B$ and $A \overline{\triangleleft} B$, where A and B are activity names from two different sets, i.e. $A \in \lambda_i(T_i)$, $B \in \lambda_j(T_j)$ and $i \neq j$.

The *validity* of atomic constraints for a given trace σ over the set of activity names $\mathcal{A} = \lambda_1(T_1) \cup \lambda_2(T_2) \cup \ldots \cup \lambda_k(T_k)$ is defined as follows:

- $\sigma \vDash A \triangleleft B \Leftrightarrow$ if B occurs in σ, then A occurs before B;
- $\sigma \vDash A \overline{\triangleleft} B \Leftrightarrow$ if A does not occur before B in σ.

When $\sigma \vDash \phi$, we say that ϕ is *valid* for σ, and σ *satisfies* ϕ. The validity of the atomic constraints has a natural interpretation.

The constraint $A \triangleleft B$ means that B should be always preceded by A, e.g. a message can be received only if it has already been sent. Thus, $A \triangleleft B$ is valid for a trace $\sigma = \ldots A \ldots B \ldots$ and is not valid for a trace $\sigma = \ldots <except\ A> \ldots B \ldots$ The constraint $A \overline{\triangleleft} B$ means that B cannot occur if A has happened before, e.g. if a message has

been already sent by mail, we should not fax it again. A trace $\sigma = \ldots\langle except\ A\rangle\ldots B\ldots$ satisfies this constraint, and a trace $\sigma = \ldots A\ldots B\ldots$ does not satisfy it. However, these atomic constraints are not negations of each other. Both $A \lhd B$ and $A \overline{\lhd} B$ are valid for a trace that does not contain B.

Now a language of interface constraints is defined by the following grammar rules:

$$Atom ::= A \lhd B \mid A \overline{\lhd} B,$$
$$\phi ::= Atom \mid \phi \vee \phi \mid \phi \wedge \phi,$$

where *Atom* is an atomic constraint, and ϕ is a constraint formula.

Validity of a constraint formula φ for a given trace σ is defined in a standard way:

$$\sigma \vDash \phi_1 \wedge \phi_2 \Leftrightarrow \sigma \vDash \phi_1 \text{ and } \sigma \vDash \phi_2,$$
$$\sigma \vDash \phi_1 \vee \phi_2 \Leftrightarrow \sigma \vDash \phi_1 \text{ or } \sigma \vDash \phi_2.$$

Let L be an event log over a set $\mathcal{A}$ of activity names, and ϕ be a constraint formula, then ϕ is valid for L iff ϕ is valid for each trace in L.

Interface formulae allow us to express different useful interaction constraints, e.g. the formula $\phi = A \overline{\lhd} B \wedge B \overline{\lhd} A$ describes a *conflict* between A and B, i.e. A and B cannot occur in the same trace.

Recall that a MAS model consists of k agent nets $N_1, N_2, \ldots, N_k$, where $N_i = (P_i, T_i, F_i, m_0^i, \lambda_i)$, and a constraint formula $\mathcal{I}$ (*interface*) with atomic constraints that defines the relations on activities of different agents.

It is easy to see that the *union* of Petri nets (considering several disjoint graphs as one disconnected graph) is also a Petri net. Thus, we can consider k agent nets as a single Petri net N. Recall that a run for a Petri net N is a sequence of activity names, corresponding to a firing sequence of N, and a trace from the related event log. Then a run of a MAS model $S = (N_1, N_2, \ldots, N_k, \mathcal{I})$ is defined as a run ρ in N satisfying $\mathcal{I}$, i.e. $\rho \vDash \mathcal{I}$.

The following proposition is the immediate consequence of the definitions.

Proposition 1: Let $S = (N_1, N_2, \ldots, N_k, \mathcal{I})$ be a MAS model, and ρ be a run in S. Then for all i the projection $\rho|_{\lambda_i(T_i)}$ on transition labels of an agent net N_i is a run in N_i.

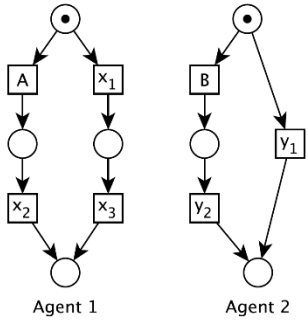


Fig. 2. A multi-agent system with two interacting systems

Consider as an example the system shown in Fig. 2 with $\mathcal{I} = A \overline{\triangleleft} B \wedge B \overline{\triangleleft} A$ meaning that A conflicts with B. Consider a run $\sigma = x_1 B y_2 x_3$ satisfying $\mathcal{I}$. Projecting σ on agent nets gives traces $x_1 x_3$ and $B y_2$, which are runs of the corresponding agent nets. This property will be further used for designing the simulation algorithm presented in the next section.

5. Simulating MAS Process Models

In this section, we describe an algorithm for simulating MAS models. It has been implemented as a ProM 6 plug-in extending GENA tool [22].

5.1 An Interface-Driven Firing Rule

A constraint formula in a MAS model defines *declarative* restrictions on the model behavior. To simulate the model behavior, we need to define operational semantics for MAS models based on a special firing rule for selecting and executing the next step in the run of the model. We call this rule an *interface-driven firing rule* to distinguish it from the standard Petri net firing rule. Naturally, this rule should be consistent with the declarative definitions of MAS model behavior.

Let $S = (N, \mathcal{I})$ be a MAS model, where a Petri net $N = (P, T, F, m_0, \lambda)$ is a union of all agent nets.

Firstly, we convert $\mathcal{I}$ to a disjunctive normal form (DNF) using standard logical laws. Then, an interface $\mathcal{I} = \bigvee C_j$ for $j = 1, 2, \ldots, n$, where $C_j = \bigwedge S_l$, and S_l is an atomic constraint for $l = 1, 2, \ldots, m$. By abuse of notation, we also denote by $\mathcal{I}$ the set of its conjuncts, and by C_j – the set of atomic constraints in a conjunct C_j.

Obviously, a trace σ satisfies $\mathcal{I}$ iff $\exists\, C_j \in \mathcal{I}$: $\sigma \vDash C_j$, i.e. it should satisfy at least one conjunct in $\mathcal{I}$. Thus, to generate a model run, we choose a conjunct C_j and fire transitions of N only if they do not violate C_j.

Then we define $T_{\mathcal{I}} \subseteq T$ to be the set of transitions involved in agent interaction, i.e. $t \in T_{\mathcal{I}}$ iff $\lambda(t)$ occurs in $\mathcal{I}$. We call transitions from $T_{\mathcal{I}}$ *interface* transitions. Independent transitions from $T \setminus T_{\mathcal{I}}$ fire according to the standard firing rule for Petri nets. The firing of interface transitions is restricted by the constraint formula. To check whether firing of a transition t violates C_j, we keep the current historical model run, i.e. a sequence of already fired activities. When a transition $t \in T_{\mathcal{I}}$ is enabled according to the standard Petri net firing rule at a current marking m, and an atomic constraint $A \triangleleft \lambda(t)$ occurs in C_j, then t is defined to be enabled only if A occurs in the current run. Similarly, if $A \overline{\triangleleft} \lambda(t)$ occurs in C_j, then t is enabled only if A does not occur in the current run. Otherwise, a transition t is enabled in the model, when it is enabled in N.

Now the *operational semantics* of a MAS model $S = (N, \mathcal{I})$, where $N = (P, T, F, m_0, \lambda)$ and $\mathcal{I} = \bigvee C_j$ for $j = 1, 2, \ldots, n$, is defined by the following procedure.

Step 1. Choose nondeterministically a conjunct C in $\mathcal{I}$.

Step 2. Start with the initial marking m_0 and ε for the current run σ.

Step 3. For a current marking m and a current run σ repeat while there are enabled transitions in N:

1) compute the set T_{ok} of all transitions enabled at m and not violating constraints from C w.r.t. σ;
2) choose nondeterministically a transition t from T_{ok};
3) fire t by changing the current marking to m', $m[t\rangle m'$, and adding $\lambda(t)$ to σ.

5.2 Event Log Generation

This subsection presents an algorithm for generating an event log by simulating behavior of a MAS model.

Let $S = (N, \mathcal{I})$ be a MAS model, where $N = (P, T, F, m_0, \lambda)$ is a Petri net, and $\mathcal{I}$ is in DNF. Firstly, for each conjunct C occurring in $\mathcal{I}$, we run (*simulate*) S to check if it is possible to obtain a trace σ satisfying C. If we cannot obtain such a trace, we exclude this conjunct. As a result, we come to a set of conjuncts $\mathcal{I}' \subseteq \mathcal{I}$, which can be actually satisfied by traces of S or an empty set if $\mathcal{I}$ cannot be satisfied by traces of S. If $\mathcal{I}' = \varnothing$, then the simulation is terminated producing an empty event log L.

That is why we can simulate S w.r.t. conjuncts occurring in $\mathcal{I}'$ only. Starting a new *iteration* of simulation, we randomly choose a conjunct from $\mathcal{I}'$ and fire transitions of N according to the interface-driven firing rule.

The end user specifies the final marking m_f, which is actually the set of sink places of agent nets. Apart from that, the log generation is regulated by the number of logs, the number of traces in a log, and by the maximum number of steps which can be executed while generating a single trace (denoted further by *maxSteps*).

Algorithm 1 is used for generating a single trace that satisfies C from $\mathcal{I}'$.

Algorithm 1. Single trace generation

```
Input: N = (P, T, F, m0, λ), I′, and mf
Output: a trace σ, s.t. σ ⊨ I′
σ ← ε; m ← m0; i ← 1; C ← pickRandomConjunct(I′)
while (i ≤ maxSteps) ∧ (m ≠ mf) do
    Tok ← findEnabledTransitions(N, m, C, σ)
    if Tok ≠ ∅ then
        t ← pickRandomTransition(Tok)
        m ← fireTransition(N, m, t)
        if λ(t) ≠ τ then
            σ ← σ + λ(t); i ← i + 1
        end
    else
        σ ← ε; break
    end
end
```

Algorithm 2 is used for finding enabled transitions, which do not violate constraints of C. Firstly, we find a set of transitions enabled at a reachable marking m according

to the standard firing rule. Secondly, if m enables interface transitions, we check whether the current run $\sigma = \lambda(w)|_{\mathcal{A}}$, s.t. $m_0[w\rangle m$, satisfies constraints of C using the interface-driven firing rule. A run σ is a trace to be recorded into an event log L.

Algorithm 2. Function findEnabledTransitions

```
Input: N = (P, T, F, m0, λ), m ∈ [m0⟩, C ∈ ℐ′, σ
Output: a set Tok of transitions enabled w.r.t. C
Tm ← stEnabledTransitions(N, m)
Tok ← Tm\Tℐ
foreach t ∈ Tm ∩ Tℐ do
    foreach S ∈ C do
        if S = X ◁ λ(t) then
            if σ = uXv then Tok ← Tok∪t
        else if S = X ◁̄ λ(t) then
            if σ≠uXv then Tok ← Tok∪t
        end
    end
end
```

We do not show here how the transition firing is implemented. It is discussed in detail in [22] where the original GENA plug-in is described.

Consider an example based on the system shown in Fig. 2. Assume $\mathcal{I} = (A \triangleleft B) \vee (y_1 \triangleleft x_1 \wedge x_2 \mathbin{\bar{\triangleleft}} y_1)$. $C = y_1 \triangleleft x_1 \wedge x_2 \mathbin{\bar{\triangleleft}} y_1$ is chosen. We are at the initial marking, i.e. $\sigma = \varepsilon$. Enabled transitions are $\{A, x_1, B, y_1\}$. However, x_1 cannot fire, since it should wait until y_1 is executed. Then B fires nondetermenistically. Subsequently, the run is $\sigma = B$, and the enabled transitions are $\{A, x_1, y_2\}$, but x_1 still cannot fire. We can choose A to fire. Then the run is $\sigma = BA$, and the enabled transitions are $\{x_2, y_2\}$ firing of which is not influenced by C. As a result, we can obtain a trace $\sigma = BAy_2x_2$ satisfying C, and the projections of σ on agent transitions, Ax_2 and By_2, are the runs of corresponding agent nets.

5.3 Experimental Simulation

We have developed the extension to the ProM[2] plug-in GENA implementing the proposed simulation algorithm and allowing users to obtain a set of event logs by simulating a given MAS model w.r.t. interaction constraints.

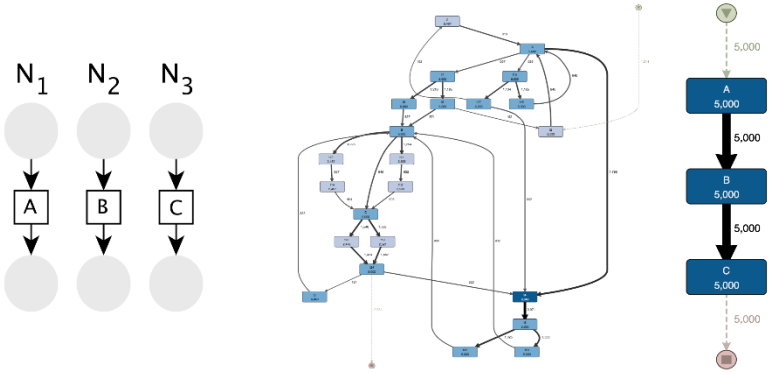
We have prepared five use cases for evaluating the proposed simulation approach. In each case, we have generated event logs with 5000 traces. In addition, we provide a "filtered" version of a generated event log w.r.t. interacting actions, s.t. it is clear whether the corresponding interface is exactly observed.

We have used Disco[3] to visualize generated event logs. Insignificant parts of agent nets are shown by shaded ovals.

[2] ProM 6 Framework page: http://www.promtools.org

[3] Fluxicon Disco page: https://fluxicon.com/disco/

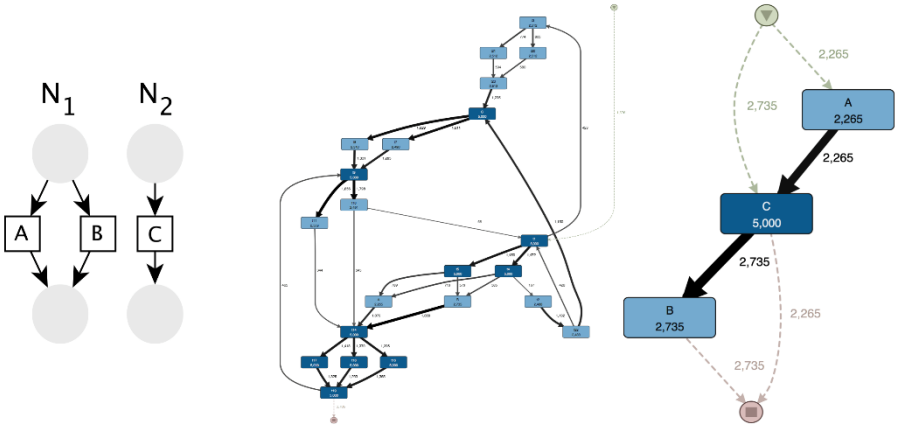
a) Sequencing: Consider a system with three interacting agents (see Fig. 3). Each agent always executes one action. We have simulated it w.r.t. the interface $\mathcal{I} = A \triangleleft B \wedge B \triangleleft C$. Intuitively, in this case each interacting agent prepares resources needed for the other agent.



(a) a system (b) an event log

Fig. 3. Sequential interaction

b) Conditional sequencing: As opposed to sequencing, conditional sequencing allows for several execution options. In this case, a system consists of two agents, one of which has two alternative branches (see Fig. 4). The interface for the conditional sequencing is as follows: $\mathcal{I} = A \triangleleft C \vee C \triangleleft B$.

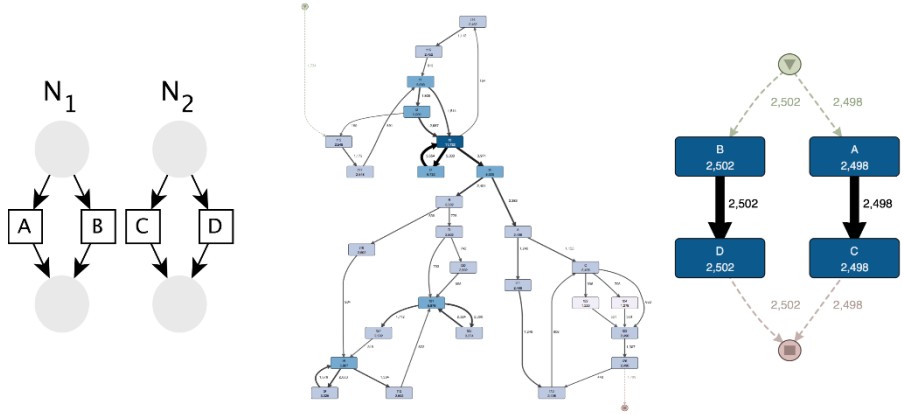


(a) a system (b) an event log

Fig. 4. Sequential interaction with options

c) Alternative interaction: The alternative interaction implies that one of two interacting agents influences the choice done by the other agent. A system consists

of two interacting agents both having two alternative branches (see Fig. 5). The interface formula for this case is as follows: $\mathcal{I} = A \triangleleft C \vee B \triangleleft D$.



(a) a system (b) an event log

Fig. 5. Alternative interaction

d) Interaction using negative constraints: Assume we have a system of two interacting agents with two alternative branches as shown in Fig. 5a. The result of simulating this system w.r.t. the interface $\mathcal{I} = A \overline{\triangleleft} C$ is shown in Fig. 6. It is clear from the simulation result that C is never preceded by A. Intuitively, negative constraints allow for a more compact way of interface construction.

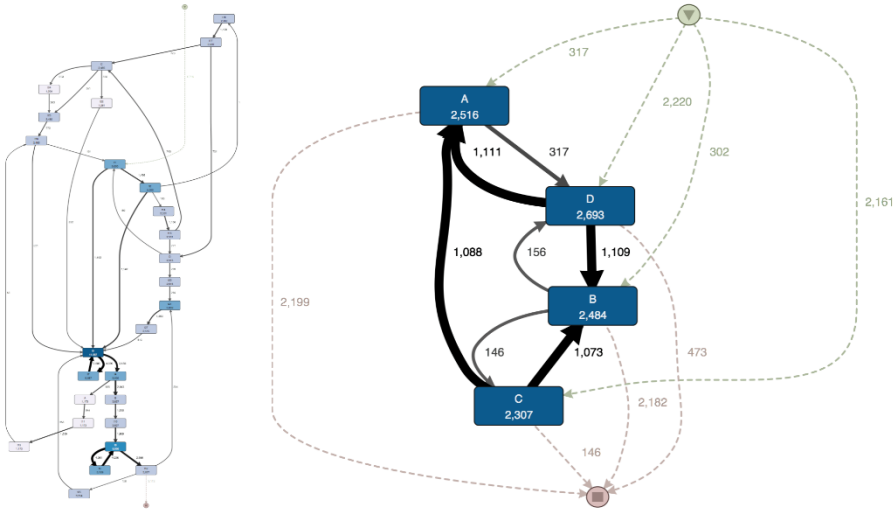
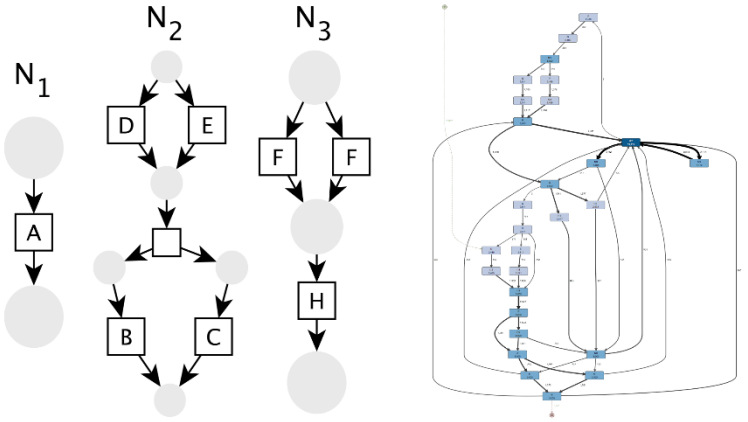


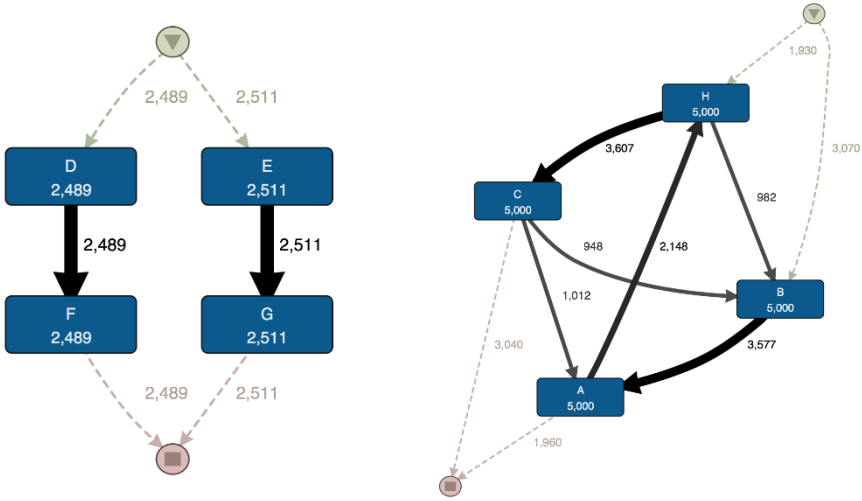
Fig. 6. Interaction using negative constraints: an event log

e) Complex interaction: In this case, we show several ways of interaction among three different agents (see Fig. 7a). For convenience, we have filtered the obtained log in two ways (see Fig. 8). We have used the following interface formula (given in a conjunctive normal form for the convenience of a reader): $\mathcal{I} = B \triangleleft A \wedge H \triangleleft C \wedge (D \triangleleft F \vee E \triangleleft G)$.



(a) a system (b) a full event log

Fig. 7. Complex interaction



(a) actions D, E, F, G (b) actions A, B, C, H

Fig. 8. Complex interaction: filtered event logs

6. Conclusion

We have proposed the new approach to model and simulate multi-agent systems using Petri nets. Independent agents are modeled as a set of labeled workflow nets, and their interaction is described using a declarative interface. The interface is constructed as a logic formula over atomic constraints describing the order of internal agent actions. This study has considered only acyclic agent interactions described by two kinds of atomic constraints, s.t. interacting activities are implemented only once. If cyclic interactions are allowed, subtler relations on interacting activities are needed to express such constraints as "*each B* should be preceded by *A*" or "*at least one B* should be preceded by *A*". This is a subject for further research.

An algorithm for simulating process models of multi-agent systems with respect to the interface has been developed. We have implemented the algorithm within the existing ProM 6 plug-in GENA and have evaluated it using five different cases of agent interactions. The experiment results show how to obtain artificial event logs by simulating process models of multi-agent systems with a finite number of asynchronously interacting agents.

References

[1]. van der Aalst W.M.P. Process Mining – Data Science in Action. Springer, Heidelberg, 2016, 467 p.

[2]. Günther C.W., van der Aalst W.M.P. Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. BPM 2007. LNCS, vol. 4714, 2007, pp. 328-343.

[3]. van der Werf J.M.E.M., van Dongen B.F., Hurkens C.A.J., Serebrenik A. Process Discovery using Integer Linear Programming. Fundamenta Informaticae, vol. 94, no. 3-4, 2009, pp. 387-412.

[4]. Weijters A.J.M.M., Ribeiro J.T.S. Flexible Heuristics Miner (FHM). In Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310-317.

[5]. Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. Process Model Discovery: A Method Based on Transition System Decomposition. ICATPN 2014. LNCS, vol. 8489, 2014, pp. 71-90.

[6]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Scalable Process Discovery with Guarantees. BPMDS 2015, EMMSAD 2015. LNBIP, vol 214, 2015, pp. 85-101.

[7]. Begicheva A.K., Lomazova I.A. Discovering high-level process models from event logs. Modeling and Analysis of Information Systems, vol. 24, no. 2, 2017, pp. 125–140..

[8]. Augusto A., Conforti R., Dumas M., La Rosa M., Maria Maggi F., Marrella A., Mecella M., Soo A. Automated Discovery of Process Models from Event Logs: Review and Benchmark. CoRR, 2017, vol. abs/1705.02288.

[9]. Rubin V.A., Mitsyuk A.A., Lomazova I.A., van der Aalst W.M.P. Process Mining can be applied to software too! In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14), 2014, pp. 1-8.

[10]. Leemans M., van der Aalst W.M.P. Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. MODELS 2015, pp. 44-53.
[11]. Leemans M., van der Aalst W.M.P., van den Brand M. Recursion Aware Modeling and Discovery for Hierarchical Software Event Log Analysis (Extended). CoRR, 2017, vol. abs/1710.09323.
[12]. Liu C., van Dongen B.F., Assy N., van der Aalst W.M.P. Component behavior discovery from software execution data. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1-8.
[13]. Davydova K.V., Shershakov S.A. Mining hybrid UML models from event logs of SOA systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 155-174. DOI: 10.15514/ISPRAS-2017-29(4)-10.
[14]. 3TU: Big software on the run. [Online]. Available: http://www.3tu-bsr.nl. Accessed: 09.06.2018.
[15]. Cabac L., Denz N. Net Components for the Integration of Process Mining into Agent-Oriented Software Engineering. Transactions on Petri Nets and Other Models of Concurrency I. LNCS, vol. 5100, 2008, pp. 86-103.
[16]. Cabac L., Knaak N., Moldt D., Rölke H. Analysis of Multi-Agent Interactions with Process Mining Techniques. MATES 2006. LNCS, vol. 4196, 2006, pp. 12-23.
[17]. Nesterov R.A., Lomazova I.A. Using Interface Patterns for Compositional Discovery of Distributed System Models. Trudy ISP RAN/Proc. ISP RAS, 2017, vol. 29, issue 4, pp. 21-38. DOI: 10.15514/ISPRAS-2017-29(4)-2.
[18]. Nesterov R.A., Lomazova I.A. Compositional Process Model Synthesis Based on Interface Patterns. TMPA 2017. CCIS, vol. 779, 2018, pp. 151-162.
[19]. Burattin A. PLG2: Multiperspective Process Randomization with Online and Offline Simulations. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 1-6.
[20]. Jouck T., Depaire B. PTandLogGenerator: A Generator for Artificial Event Data. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 23-27.
[21]. Yan Z., Dijkman R.M., Grefen P. Generating process model collections. Software and System Modeling, 2017, vol. 16, issue 4, pp. 979-995.
[22]. Shugurov I.S., Mitsyuk A.A. Generation of a Set of Event Logs with Noise. In Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88-95. DOI: 10.15514/SYRCOSE-2014-8-13.
[23]. Mitsyuk A.A., Shugurov I.S., Kalenkova A.A., van der Aalst W.M.P. Generating event logs for high-level process models. Simulation Modelling Practice and Theory, vol. 74, 2017, pp. 1-16.
[24]. de Medeiros A.K.A., Günther C.W. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In Proceedings of CPN 2005. DAIMI, vol. 576, 2005, pp. 177-190.
[25]. Di Ciccio C., Luca Bernardi M., Cimitile M., Maria Maggi F. Generating Event Logs Through the Simulation of Declare Models. EOMAS 2015. LNBIP, vol. 231, 2015, pp. 20-36.

Симуляция поведения мультиагентных систем с ациклически взаимодействующим агентами

[1, 2] *Р.А. Нестеров <rnesterov@hse.ru, r.nesterov@campus.unimib.it>*
[1] *А.А. Мицюк <amitsyuk@hse.ru>*
[1] *И.А. Ломазова <ilomazova@hse.ru>*
[1] *Национальный исследовательский университет «Высшая школа экономики», 101000, Россия, г. Москва, ул. Мясницкая, д. 20.*
[2] *Департамент информатики, систем и коммуникаций, Миланский университет-Бикокка, 20126, Италия, г. Милан, Viale Sarca 336 – Edificio U14*

Аннотация. В работе предложен подход для моделирования и симуляции поведения мультиагентных систем (МАС) с применением сетей Петри. МАС представляется как конечное множество сетей потоков работ. Асинхронные взаимодействия агентов описываются с помощью интерфейса, который определяется логической формулой над множеством атомарных ограничений. Эти ограничения задают порядок выполнения внутренних действий агентов. В статье рассматриваются только ациклические взаимодействия агентов. Также был разработан алгоритм симуляции поведения МАС с учетом ограничений взаимодействия агентов. Алгоритм реализован в виде подключаемого модуля для инструмента ProM 6. Предложенный подход может быть использован для оценки качества алгоритмов извлечения процессов (process discovery) с точки зрения характеристик получаемых моделей процессов.

Ключевые слова: сети Петри; мультиагентные системы; взаимодействие; интерфейс; симуляция; журналы событий

DOI: 10.15514/ISPRAS-2018-30(3)-20

Для цитирования: Нестеров Р.А., Мицюк А.А., Ломазова И.А. Симуляция поведения мультиагентных систем с ациклически взаимодействующими агентами. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 285-302 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-20

Список литературы

[1]. van der Aalst W.M.P. Process Mining – Data Science in Action. Springer, Heidelberg, 2016, 467 p.
[2]. Günther C.W., van der Aalst W.M.P. Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. BPM 2007. LNCS, vol. 4714, 2007, pp. 328-343.
[3]. van der Werf J.M.E.M., van Dongen B.F., Hurkens C.A.J., Serebrenik A. Process Discovery using Integer Linear Programming. Fundamenta Informaticae, vol. 94, no. 3-4, 2009, pp. 387-412.

[4]. Weijters A.J.M.M., Ribeiro J.T.S. Flexible Heuristics Miner (FHM). In Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310-317.
[5]. Kalenkova A.A., Lomazova I.A., van der Aalst W.M.P. Process Model Discovery: A Method Based on Transition System Decomposition. ICATPN 2014. LNCS, vol. 8489, 2014, pp. 71-90.
[6]. Leemans S.J.J., Fahland D., van der Aalst W.M.P. Scalable Process Discovery with Guarantees. BPMDS 2015, EMMSAD 2015. LNBIP, vol 214, 2015, pp. 85-101.
[7]. Begicheva A.K., Lomazova I.A. Discovering high-level process models from event logs. Modeling and Analysis of Information Systems, vol. 24, no. 2, 2017, pp. 125–140..
[8]. Augusto A., Conforti R., Dumas M., La Rosa M., Maria Maggi F., Marrella A., Mecella M., Soo A. Automated Discovery of Process Models from Event Logs: Review and Benchmark. CoRR, 2017, vol. abs/1705.02288.
[9]. Rubin V.A., Mitsyuk A.A., Lomazova I.A., van der Aalst W.M.P. Process Mining can be applied to software too! In Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14), 2014, pp. 1-8.
[10]. Leemans M., van der Aalst W.M.P. Process mining in software systems: Discovering real-life business transactions and process models from distributed systems. MODELS 2015, pp. 44-53.
[11]. Leemans M., van der Aalst W.M.P., van den Brand M. Recursion Aware Modeling and Discovery for Hierarchical Software Event Log Analysis (Extended). CoRR, 2017, vol. abs/1710.09323.
[12]. Liu C., van Dongen B.F., Assy N., van der Aalst W.M.P. Component behavior discovery from software execution data. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), 2016, pp. 1-8.
[13]. Davydova K.V., Shershakov S.A. Mining hybrid UML models from event logs of SOA systems. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 155-174. DOI: 10.15514/ISPRAS-2017-29(4)-10.
[14]. 3TU: Big software on the run. [Online]. Available: http://www.3tu-bsr.nl. Accessed: 09.06.2018.
[15]. Cabac L., Denz N. Net Components for the Integration of Process Mining into Agent-Oriented Software Engineering. Transactions on Petri Nets and Other Models of Concurrency I. LNCS, vol. 5100, 2008, pp. 86-103.
[16]. Cabac L., Knaak N., Moldt D., Rölke H. Analysis of Multi-Agent Interactions with Process Mining Techniques. MATES 2006. LNCS, vol. 4196, 2006, pp. 12-23.
[17]. Nesterov R.A., Lomazova I.A. Using Interface Patterns for Compositional Discovery of Distributed System Models. Trudy ISP RAN/Proc. ISP RAS, 2017, vol. 29, issue 4, pp. 21-38. DOI: 10.15514/ISPRAS-2017-29(4)-2.
[18]. Nesterov R.A., Lomazova I.A. Compositional Process Model Synthesis Based on Interface Patterns. TMPA 2017. CCIS, vol. 779, 2018, pp. 151-162.
[19]. Burattin A. PLG2: Multiperspective Process Randomization with Online and Offline Simulations. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 1-6.
[20]. Jouck T., Depaire B. PTandLogGenerator: A Generator for Artificial Event Data. BPMD 2016. CEUR Workshop Proceedings, vol. 1789, 2016, pp. 23-27.
[21]. Yan Z., Dijkman R.M., Grefen P. Generating process model collections. Software and System Modeling, 2017, vol. 16, issue 4, pp. 979-995.

[22]. Shugurov I.S., Mitsyuk A.A. Generation of a Set of Event Logs with Noise. In Proceedings of the 8th Spring/Summer Young Researchers Colloquium on Software Engineering (SYRCoSE 2014), 2014, pp. 88-95. DOI: 10.15514/SYRCOSE-2014-8-13.

[23]. Mitsyuk A.A., Shugurov I.S., Kalenkova A.A., van der Aalst W.M.P. Generating event logs for high-level process models. Simulation Modelling Practice and Theory, vol. 74, 2017, pp. 1-16.

[24]. de Medeiros A.K.A., Günther C.W. Process Mining: Using CPN Tools to Create Test Logs for Mining Algorithms. In Proceedings of CPN 2005. DAIMI, vol. 576, 2005, pp. 177-190.

[25]. Di Ciccio C., Luca Bernardi M., Cimitile M., Maria Maggi F. Generating Event Logs Through the Simulation of Declare Models. EOMAS 2015. LNBIP, vol. 231, 2015, pp. 20-36.

On the model checking of finite state transducers over semigroups

[1] *A.R. Gnatenko<gnatenko.cmc@gmail.com>*
[2] *V.A. Zakharov<zakh@cs.msu.su>*
[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2] *National Research University High School of Economics,*
20, Myasnitskaya str., Moscow, 101000, Russia

Abstract. Sequential reactive systems represent programs that interact with the environment by receiving signals or requests and react to these requests by performing operations with data. Such systems simulate various software like computer drivers, real-time systems, control procedures, online protocols etc. In this paper, we study the verification problem for the programs of this kind. We use finite state transducers over semigroups as formal models of reactive systems. We introduce a new specification language LP-CTL* to describe the behavior of transducers. This language is based on the well-known temporal logic CTL* and has two distinguished features: 1) each temporal operator is parameterized with a regular expression over input alphabet of the transducer, and 2) each atomic proposition is specified by a regular expression over the output alphabet of the transducer. We develop a tabular algorithm for model checking of finite state transducers over semigroups against LP-CTL* formulae, prove its correctness, and estimate its complexity. We also consider a special fragment of LP-CTL* language, where temporal operators are parameterized with regular expressions over one-letter alphabet, and show that this fragment may be used to specify usual Kripke structures, while it is more expressive than usual CTL*.

Keywords: reactive program**;** transducer**;** verification**;** model checking**;** temporal logic**;** finite state automaton**;** regular language

DOI: 10.15514/ISPRAS-2018-30(3)-21

For citation: Gnatenko A.R., Zakharov V.A. On the model checking of finite state transducers over semigroups. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 303-324. DOI: 10.15514/ISPRAS-2018-30(3)-21

1. Introduction

Finite state machines are widely used in the field of computer science and formal methods for various purposes. While finite automata represent regular sets, *transducers* stand for regular (or, rational) *relations* and, therefore, can serve as models of programs and algorithms that operate with input and output data. For

example, transducers are used as formal models in software engineering to represent numerous algorithms, protocols and drivers that manipulate with strings, dataflows, etc. [1, 15, 25].

By extending the concept of ordinary transducers, we build a new formal model for sequential reactive systems. These systems are software programs or hardware devices that receive requests (control signals, commands) from the environment and perform in response some manipulations (actions, transformations) with data, interactions with the environment, mechanical movements, etc. While the flow of requests can be represented by finite or infinite words in some fixed alphabet, the sequence of actions of the system needs a more sophisticated interpretation. The key point here is that different sequences of actions may bring a computing system to the same result. To capture this effect the collection of actions performed by a reactive system can be viewed as a generating set of some algebraic structure (e.g. semigroup, group, ring, etc.) and particular algebraic properties of basic actions should be taken into account when choosing adequate formal models for this class of information processing systems. Let us illustrate this consideration by several examples.

- A network switch with several input and output ports. A switch is a device, which receives data packets on its input port, modifies their heads and commutes them to one of the output ports. Once received a special control signal, this switch changes its packet-forwarding table and, thus, its behaviour. Since packets from different flows may be processed in any order, the switch can be modeled by a transducer, which operates over a free partially commutative semigroup, or a trace monoid. Trace monoids are commonly used as an algebraic foundation of concurrent computations and process calculi (see, e.g., [9]).
- A real-time device that control the operation of some industrial equipment (say, a boiling system). Such device receives data from temperature and pressure sensors and switches some processes on and off according to its instructions and the current state of the system. It seems reasonable that for some actions the order of their implementation is not important (routine actions), while others must follow in a strictly specified order (e.g. an execution of some complex operation). Moreover, there are also actions which bring system to certain predefined operation mode (set-up actions). These actions are implemented in the emergency situations. A partially commutative semigroup with right-zero elements $\mathbf{0}$ which satisfy the equalities $x\mathbf{0} = \mathbf{0}$ for every element x provides an adequate interpretation for such operations.
- A system supervisor that maintains a log-file. For each entry its date and time is recorded in the file and there is no way to delete entries from the log – only to append it. Thus, for any two basic actions (record operations to the log-file) it is crucial in which order they are performed and such a supervisor can be modeled by a transducer over a free semigroup. Verification techniques for such reactive

systems are considered in [17]; this is the main topic of this paper as well.

- A radio-controlled robot, that moves on a surface. It can make one step moves in any of direction. When it receives a control signal in a state q' it must choose and carry out a sequence of steps and enter to the next state q''. At some distinguished state q_f the robot reports its current location. Movements of the robot may be regarded as basic actions, and the simplest model of computation which is suitable for analyzing a behaviour of this robot is a nondeterministic finite state transducer operating on a free Abelian group of rank 2.

To construct a reliable system or network it is crucial for its components to have a correct behaviour. For example, a network switch must process received data packets within a specified number of execution steps and the boiling system should never be overheated, that is, will never remain for a long time in a particular condition without appropriate responses from the control device. By using finite state transducers as formal models of reactive systems, one can develop verification algorithms for these models to solve such problems as equivalence checking, deductive verification or model checking.

The study of the equivalence checking problem for classical transducers began in the early 60s. It was established that the equivalence checking problem for non-deterministic transducers is undecidable [13] even over 1-letter input alphabet [16]. However, the undecidability displays itself only in the case of unbounded transductions when an input word may have arbitrary many images. The equivalence checking problem was shown to be decidable for deterministic [4], functional (single-valued) [5, 19], and k-valued transducers [6, 26]. In a series of papers [20-22] techniques for checking bounded valuedness, k-valuedness and equivalence of finite state transducers over words were developed. Recently in [29] equivalence checking problem was shown to be decidable for finite state transducers that operate over finitely generated semigroups embeddable in decidable groups.

There are also papers where equivalence checking problem for transducers is studiedin the framework of program verification. The authors of [23] proposed models of communication protocols as finite state transducers operating on bit strings. They set up the verification problem as equivalence checking between the protocol transducer and the specification transducer. The authors of [25] extended finite state transducers with symbolic alphabets, which are represented as parametric theories. They showed that a number of classical problems for extended transducers, including equivalence checking problem, are decidable modulo underlying theories. In [1] a model of streaming transducers was proposed for programs that access and modify sequences of data items in a single pass. It was shown that a number of verification problems such as equivalence checking, assertion checking, and checking correctness with respect to pre/post conditions, are decidable for this program model.

Meanwhile, very few papers on the model checking problem for transducers are known. Transducers can be conveniently used as auxiliary means in regular model

checking of parameterized distributed systems where configurations are represented as words over a finite alphabet. In such models, a transition relation on these configurations may be regarded as a rational relation and, thus, it may be specified by finite state transducers (see [7, 28]). In these papers, finite state transducers just play the role of verification instrument, but not an object of verification. However, as far as we know, a deeper investigation of the model checking problem for the reactive systems represented by transducers has not yet been carried out. We think that this is due the following main reason. A transducer is a model of computation which, given an input word, computes an output word. The letters of input and output alphabets can be regarded as valuations (tuples of truth values) of some set of basic predicates. Therefore, a transducer can be viewed as some special representation of a labeled transition system (Kripke structure) (see [2]). From this viewpoint model checking problem for finite state transducers conforms well to standard model checking scheme for finite structures, and, hence, it is not worthy of any particular treatment.

However, our viewpoint is quite different. Transducer is a more complex model of computation than a finite state automaton (transition systems). Its behaviorism characterized by the correspondence between input and output words. A typical property of such behaviour to be checked is whether for every (some) input word from a given pattern a transducer outputs a word from another given pattern. Therefore, when formally expressing the requirements of this kind one needs not only temporal operators to specify an order in which events occur but also some means to refer to such patterns. Conventional Temporal Logics like $\boldsymbol{LTL}$ or $\boldsymbol{CTL}$ are not sufficient in this case; they should be modified in such a way as to acquire an ability to express correspondences between the sets (languages) of input words and the sets (languages) of output words. This could be achieved by supplying temporal operators with patterns as parameters. Every such pattern is a formal description of a language L over an input alphabet$\mathcal{C}$; automata, formal grammars, regular expressions, language equations are suitable for this purpose. The basic properties of output words can be also represented by languages over an output alphabet. Then, for instance, an expression $\boldsymbol{G}_L P$ can be understood as the requirement that for every input word w from the language L the output word h of a transducer belongs to the language P.

The advantages of this approach are twofold. On the one hand, such extensions of Temporal Logics make it possible to express explicitly relationships between input and output words and specify thus desirable behaviours of transducers. On the other hand, it can be hoped that such extensions could rather easily assimilate some well-known model checking techniques (see [3, 8]) developed for conventional Temporal Logics. The first attempt to implement this approach was made in [17]. The authors of this paper introduced an $\boldsymbol{\mathcal{LP}\text{-}LTL}$ specification language based on $\boldsymbol{LTL}$ temporal logic and developed a checking procedure for transducers over free monoids against specifications from $\boldsymbol{\mathcal{LP}\text{-}LTL}$. It was shown that this procedure has double exponential time complexity.

In this paper we continue this line of research and "raise" the specification language introduced in [17] to the level of $\mathcal{LP}$-$\mathbf{CTL}^*$. We will focus only on one task related to the use of new logic for the verification of reactive systems, namely, the development of a general model checking algorithm for finite state transducers against specifications in $\mathcal{LP}$-$\mathbf{CTL}^*$. Such issues as expressive power of $\mathcal{LP}$-$\mathbf{CTL}^*$, complexity of model checking and satisfiability checking problems, the influence of types of languages used as parameters and basic predicates in $\mathcal{LP}$-$\mathbf{CTL}^*$ on decidability and complexity of model checking problem remain topic of our further research and will be covered in our subsequent works. We also leave beyond this work a number of applied questions, which are worthy of consideration in a separate paper. For example, it is important to understand to what extent the already developed model checking tools can be adapted to the new temporal logic. And, of course, in the future we will have a well-chosen series of examples that illustrate the new possibilities of using $\mathcal{LP}$-$\mathbf{CTL}^*$to describe the behavior of reactive systems.

The paper is organized as follows. In Section 2, we define the concept of finite state transducer over semigroup as a formal model of sequential reactive systems (see [29]) and in Section 3, we describe the syntax and the semantics of $\mathcal{LP}$-$\mathbf{CTL}^*$ as a formal language for specifying behaviour of sequential reactive systems. In Section 3 we also set up formally model checking problem for finite state transducers against $\mathcal{LP}$-$\mathbf{CTL}^*$ formulae. In Section 4, we present an $\mathcal{LP}$-$\mathbf{CTL}^*$ model checking algorithm for the case when parameters of temporal operators and basic predicates are regular languages represented by finite state automata. The model checking algorithm we designed has time complexity which is linear of the size of a transducer but exponential of the size of $\mathcal{LP}$-$\mathbf{CTL}^*$ formula. This complexity estimate is in contrast to the case of conventional $\mathbf{CTL}$ model checking: its time complexity is linear of both the size of a model and the size of a $\mathbf{CTL}$ formula. To explain this effect in Section 5 we show how $\mathcal{LP}$-$\mathbf{CTL}^*$ formulae can be also checked on the conventional Kripke structures. Finally, we compare $\mathcal{LP}$-$\mathbf{CTL}^*$ with some other known extensions Temporal Logics and discuss some topics for further research.

2. Finite state transducers as models of reactive systems

In this section, we introduce a Finite State Transducer as a formal model of a reactive computing system which receives control signals from the environment and reacts to these signals by performing operations with data.

Let $\mathcal{C}$ be a finite set of *signals*. Finite words over $\mathcal{C}$ are called *signal flows*; the set of all signal flows is denoted by$\mathcal{C}^*$. Given a pair of signal flows u and v we write uv for their concatenation, and denote by ε the empty flow.

Let $\mathcal{A} = \{a_1, \dots, a_n\}$ be a finite set of elements called *basic actions*; these actions stand for the elementary operations performed by a reactive system. Finite words over $\mathcal{A}$ are called *compound actions*; they denote sequential compositions of basic actions. Since different sequences of basic actions could produce the same result,

one may interpret compound actions over a semigroup $(S,\ e,\ \circ)$ generated by a set of basic actions $\mathcal{A}$. The elements of S are called *data states*. Every compound action $h = a_{i_1}a_{i_2}\ldots a_{i_k}$ is evaluated by the data state $[h] = [a_{i_1}] \circ [a_{i_2}] \circ \ldots \circ [a_{i_k}]$. For example, if a reactive system just keeps a track of input requests by adding certain records to a log-file then a free semigroup will be suitable for interpretation of these operations. But when a robot moves on a 2-dimensional surface then the actions (movements) performed by this robot may be regarded as generating elements of Abelian group G of rank 2, and the positions on the surface occupied by this robot can be specified by the elements from G. In this paper we restrict ourselves to the consideration of free semigroups when $[h] = h$ holds for every compound action h, and $\circ$ is the word concatenation operation.

Let $\mathcal{C}$ be a set of signals and $\mathcal{A}$ be a set of basic actions that are interpreted over a semigroup $(S, e, \circ)$. Then a Finite State Transducer (in what follows, FST) is a quintuple $\Pi = (Q,\ \mathcal{C}, \mathcal{A}, q_{init}, T)$, where

- Q is a finite set of *control states*;
- $q_{init} \in Q$ is an *initial control state*;
- $T \subseteq Q \times \mathcal{C} \times Q \times \mathcal{A}^*$is a finite *transition relation*.

Each tuple (q', c, q'', h) in T is called a *transition*: when a transducer is in a control state q' and receives a signal c, it changes its state to q'' and performs a compound action h. We denote such transition by $q' \xrightarrow{c,h} q''$. A *run* of a FST Π is any finite sequence of transitions

$$q_1 \xrightarrow{c_1,h_1} q_2 \xrightarrow{c_2,h_2} q_3 \xrightarrow{c_3,h_3} \cdots \xrightarrow{c_n,h_n} q_{n+1};$$

this run transduces a signal flow $w = c_1c_2\ldots c_n$ into a data state $[h_1h_2\ldots h_n]$.

The behaviour of a FST $\Pi = (Q,\ \mathcal{C}, \mathcal{A}, q_{init}, T)$ over a semigroup (data space) $(S, e, \circ)$ is presented formally by a *transition system* $TS(\Pi, S) = (D, \mathcal{C}, d_{init}, \mathcal{T})$, where

- $D = Q \times S$ is (in general case, infinite) set of *states of computation*,
- $d_{init} = (q_{init}, e)$ is the *initial state*, and
- $\mathcal{T} \subseteq D \times \mathcal{C} \times D$ is a *transition relation* such that for every states of computation $d' = (q', s'), d'' = (q'', s'')$ and every signal $c \in \mathcal{C}$ the relationship

  $$(d', c, d'') \in \mathcal{T} \ \Rightarrow\ \exists h \in \mathcal{A}^*(q', c, q'', h) \in T \text{ and } s'' = s' \circ [h]$$

  holds.

As usual, a transition $(d', c, d'') \in \mathcal{T}$ is denoted by $d' \xrightarrow{c} d''$.

A *trajectory* in a transition system $TS(\Pi, S)$ is a pair $tr = (d_0, \alpha)$, where $d_0 \in D$ and $\alpha = (c_1, d_1), (c_2, d_2), \ldots, (c_i, d_i), \ldots$ is a sequence of pairs (c_i, d_i) such that $d_{i-1} \xrightarrow{c_i} d_i$ holds for every $i, i \geq 1$. A trajectory represents a possible scenario of a behaviour of a sequential reactive system: when receiving a signal flow

$c_1, c_2, \dots, c_i, \dots$ the reactive system performs a sequence of basic actions h and follows sequentially via the states of computation $d_1, d_2, \dots, d_i, \dots$. By $tr|^i$ we mean the trajectory$(d_i, \alpha|^i)$, where $\alpha|^i = (c_{i+1}, d_{i+1}), (c_{i+2}, d_{i+2}), \dots$ is a suffix of α, respectively.

3. $\mathcal{LP}$-CTL^* specification language

When designing sequential reactive systems one should be provided with a suitable formalism to specify the requirements for their desirable behaviour. For example, one may expect that

- a mobile robot, receiving an equal number of control signals "go_left" and "go_right", will always return to its original position,
- a network switch will never commute data packets from different packet flows into the same output buffer,
- it is not possible for the interrupt service routine to complete the processing of one interrupt before it receives a request to handle another.

These and many other requirements which refer to the correspondences between control flows and compound actions in the course of FST runs can be specified by means of Temporal Logics. When choosing a suitable temporal logic as a formal specification language of FST behaviours one should take into account two principal features of our model of sequential reactive systems:

- since a FST operates over a data space which is semigroup, the basic predicates must be interpreted over semigroups as well, and
- since a behaviour of a FST depends not on the time flow itself but on a signal flow which it receives as an input, temporal operators must be parameterized by certain descriptions of admissible signal flows.

To adapt traditional temporal logic formalism to these specific features of FST behaviours the authors of [17] introduced a new variant of Linear Temporal Logic (***LTL***). We assume that in general case one may be interested in checking the correctness of FST's responses to arbitrary set of signal flows. Every set of control flows may be regarded as a language over the alphabet $\mathcal{C}$ of signals. Therefore, it is reasonable to supply temporal operators ("globally" $\boldsymbol{G}$, "eventually" $\boldsymbol{F}$, etc.) with certain descriptions of such languages as parameters. In more specific cases we may confine ourselves with considering only a certain family of languages (finite, regular, context-free, etc.) $\mathcal{L}$ used as parameters of temporal operators. These languages will be called *environment behaviour patterns*.

A reactive system performs finite sequences of basic actions in response to control signals from the environment and thus follows in the course of its run via a sequence of data states, which are elements of a semigroup $(S, e, \circ)$, Therefore, basic predicates used in ***LTL*** formulae may be viewed as some sets of data states S', $S' \subseteq S$. These sets can be also specified in language-theoretic fashion.

Any language P over the alphabet of basic actions $\mathcal{A}$ corresponds to a predicate (set of data states) $S_P = \{[h] \mid h \in P\}$. As in the case of environment behaviour patterns we may distinguish a certain class $\mathcal{P}$ of languages and use them as specifications of basic predicates. When these languages are used as parameters in temporal formulae then it will be assumed that they are defined constructively by means of automata, grammars, Turing machines, etc.

Thus, we arrive at the concept of $\boldsymbol{\mathcal{LP}}$-variants of Temporal Logics. In [17] the syntax and semantics of $\boldsymbol{\mathcal{LP}}$-**LTL** was studied in some details in the case when both environment behaviour patterns and basic predicates are regular languages presented by finite automata. In this paper we make one step further and extend the concept of $\boldsymbol{\mathcal{LP}}$-variants of Temporal Logics to $\boldsymbol{CTL}^*$. Select an arbitrary family of environment behaviour patterns $\mathcal{L}$ and a family of basic predicates $\mathcal{P}$. The set of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}^*$ formulae consists of *state formulae* and *trajectory formulae*, which are defined as follows:

- each basic predicate $P \in \mathcal{P}$ is a state formula;
- if φ_1, φ_2 are state formulae then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$ and $\varphi_1 \vee \varphi_2$ are state formulae;
- if ψ is a trajectory formula then $\boldsymbol{A}\psi$ and $\boldsymbol{E}\psi$ are state formulae;
- if φ is a state formula then φ is a trajectory formula;
- if ψ_1, ψ_2 are trajectory formulae then $\neg\psi_1$, $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ are trajectory formulae;
- if φ, φ_1, φ_2 are state formulae, $c \in \mathcal{C}$, and $L \in \mathcal{L}$ then $\boldsymbol{X}_c\varphi$, $\boldsymbol{Y}_c\varphi$, $\boldsymbol{F}_L\varphi$, $\boldsymbol{G}_L\varphi$ and $\varphi_1 \boldsymbol{U} \varphi_2$ are trajectory formulae.

The specification language $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}^*$ is the set of all state formulae constructed as defined above.

Now we introduce the semantics of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}^*$ formulae. These formulae are interpreted over transition systems. Let $M = TS(\Pi, S)$ be a transition system, d be a state of computation in this system, and tr be a trajectory in M. Then for every state formula φ we write $M, d \vDash \varphi$ to denote the fact that the assertion φ is true in the state d of M, and for every trajectory formula ψ we write $M, tr \vDash \psi$ to denote the fact that the assertion ψ holds for the trajectory tr in M.

In the definition below it is assumed that M is a transition system, $d = (q, s)$ is a state of computation in M, and $tr = (d_0, \alpha)$ is a trajectory in M such that $\alpha = (c_1, d_1), (c_2, d_2), \ldots, (c_i, d_i), \ldots$. We define the satisfiability relation $\vDash$ by induction on the height of formulae:

- $M, d \vDash P \Leftrightarrow s \in P$;
- $M, d \vDash \neg\varphi \Leftrightarrow$ it is not true that $M, d \vDash \varphi$;
- $M, d \vDash \varphi_1 \wedge \varphi_2 \Leftrightarrow M, d \vDash \varphi_1$ and $M, d \vDash \varphi_2$;
- $M, d \vDash \boldsymbol{E}\varphi \Leftrightarrow$ there exists a trajectory $tr' = (d, \alpha')$ in M such that $M, tr' \vDash \varphi$;

- $M, d \vDash \boldsymbol{A}\varphi \iff$ for any trajectory $tr' = (d, \alpha')$ in M it is true that $M, tr' \vDash \varphi$;
- if φ is a state formula then $M, tr \vDash \varphi \iff M, d_0 \vDash \varphi$;
- $M, tr \vDash \neg \psi \iff$ it is not true that $M, tr \vDash \psi$;
- $M, tr \vDash \psi_1 \wedge \psi_2 \iff M, tr \vDash \psi_1$ and $M, tr \backslash models\ \psi_2$;
- $M, tr \vDash \boldsymbol{X}_c\varphi \iff c = c_1$ and $M, d_1 \vDash \varphi$;
- $M, tr \vDash \boldsymbol{Y}_c\varphi \iff$ either $c \neq c_1$, or $M, d_1 \vDash \varphi$;
- $M, tr \vDash \boldsymbol{F}_L\varphi \iff \exists\, i \geq 0 : c_1c_2 \ldots c_i \in L$ and $M, tr|^i \vDash \varphi$;
- $M, tr \vDash \boldsymbol{G}_L\varphi \iff \forall\, i \geq 0 :$ if $c_1c_2 \ldots c_i \in L$ then $M, tr|^i \vDash \varphi$;
- $M, tr \vDash \varphi\, \boldsymbol{U}_L\psi \iff \exists\, i \geq 0 : c_1c_2 \ldots c_i \in L, M, tr|^i \vDash \psi$ and $\forall\, j, 0 \leq j < i,$ if $c_1c_2 \ldots c_i \in L$ then $M, tr|^j \vDash \varphi$.

Observe, that operators $\boldsymbol{X}_c$ and $\boldsymbol{Y}_c$, as well as $\boldsymbol{F}_L$ and $\boldsymbol{G}_L$, are dual to each other:

Proposition 1. *For any $\mathcal{LP}$-$\mathcal{CTL}^*$ formula φ, any $c \in \mathcal{C}$ and any $L \in \mathcal{L}$, and for an arbitrary trajectory tr in M*

- $tr \vDash \boldsymbol{X}_c\varphi \iff tr \vDash \neg \boldsymbol{Y}_c \neg \varphi$,
- $tr \vDash \boldsymbol{F}_L\varphi \iff tr \vDash \neg \boldsymbol{G}_L \neg \varphi$.

As usual, other Boolean connectives like $\vee$, $\rightarrow$, $\equiv$ may be defined by means of $\neg$ and $\wedge$. Some other $\boldsymbol{CTL}^*$ operators like, for example, **R** (release) or $\boldsymbol{W}$ (weak until) may be parametrized by environmental behaviour patterns in the same fashion.

The model checking problem we deal with is that of checking, given a finite state transducer Π operating over a semigroup $(S, e, \circ)$, and an $\mathcal{LP}$-$\mathcal{CTL}^*$ formula φ, whether $TS(\Pi, S), d_{init} \vDash \varphi$ holds. When a semigroup is fixed then we use a brief notation $\Pi \vDash \varphi$.

4. Model checking against $\mathcal{LP}$-$\mathcal{CTL}^*$ specifications

In this paper, we discuss only the most simple case of model checking problem for finite state transducers against $\mathcal{LP}$-$\mathcal{CTL}^*$ formulae when

- the semigroup $(S, \circ, e)$ the transducers operate over is a *free monoid*, which means that S is the set of all finite words in the alphabet $\mathcal{A}$, the binary operation $\circ$ is concatenation of words, and the neutral element e is the empty word ε;
- the family of environment behaviour patterns $\mathcal{L}$ is the family of regular languages in the alphabet $\mathcal{C}$;
- all basic predicates in $\mathcal{P}$ are specified by regular languages in the alphabet $\mathcal{A}$.

All regular languages used as environment behaviour patterns and basic predicate specifications are defined by means of deterministic finite state automata (DFAs). Therefore, the size of a $\mathcal{LP}$-$\mathcal{CTL}^*$ formula is the number of Boolean connectives and temporal operators occurred in φ plus the total size of automata used in φ to specify environment behaviour patterns and basic predicates.

Let us first describe a model checking algorithm for $\mathcal{LP}$-$\mathcal{CTL}$ fragment of $\mathcal{LP}$-$\mathcal{CTL}^*$, which consists of all $\mathcal{LP}$-$\mathcal{CTL}^*$ formulae such that every temporal operator $\mathbf{X}_c$, $\mathbf{Y}_c$, $\mathbf{F}_L$, $\mathbf{G}_L$ $\mathbf{U}_L$ is immediately preceded by a trajectory quantifier **E** or **A**. In our algorithm, we involve an explicit iterative model checking techniques for the ordinary $\mathcal{CTL}$ (see [8, 10]). Following this approach satisfiability checking of a formula φ in a state d of a model M is reduced to satisfiability checking of the largest subformulae of φ in the state d and in the neighboring states of M. In other words, a model checking procedure incrementally labels all states of a model by those subformulae of φ which are satisfied in these states.

Let $\Pi = (Q, \mathcal{C}, \mathcal{A}, q_{init}, T)$ be a finite state transducer over the free semigroup $(\mathcal{A}^*, \cdot, \varepsilon)$ and let φ be an $\mathcal{LP}$-$\mathcal{CTL}$ formula. There are five pairs of coupled $\mathcal{LP}$-$\mathcal{CTL}$ temporal operators: $\mathbf{AX}_c$ and $\mathbf{EX}_c$, $\mathbf{AY}_c$ and $\mathbf{EY}_c$, $\mathbf{AF}_L$ and $\mathbf{EF}_L$, $\mathbf{AG}_L$ and $\mathbf{EG}_L$, $\mathbf{AU}_L$ and $\mathbf{EU}_L$. As in the case of "ordinary" $\mathcal{CTL}$ (see), each of these couple can be expressed in terms of four main coupled operators $\mathbf{EX}_c$, $\mathbf{EY}_c$, $\mathbf{EG}_L$ and $\mathbf{EU}_L$:

Proposition 2. *For every formula φ the following equalities hold*

1. $\models \mathbf{AX}_c\varphi \equiv \neg\mathbf{EY}_c\neg\varphi$,
2. $\models \mathbf{AY}_c\varphi \equiv \neg\mathbf{EX}_c\neg\varphi$,
3. $\models \mathbf{AF}_L\varphi \equiv \neg\mathbf{EG}_L\neg\varphi$,
4. $\models \mathbf{EF}_L\varphi \equiv \mathbf{E}[true\ \mathbf{U}_L\varphi]$,
5. $\models \mathbf{AG}_L\varphi \equiv \neg\mathbf{EF}_L\neg\varphi$,
6. $\models \mathbf{A}[\varphi\ \mathbf{U}_L\psi] \equiv \neg\mathbf{E}[\neg\psi\ \mathbf{U}_L\ (\neg\varphi \wedge \neg\psi)] \wedge \neg\mathbf{EG}_L\neg\psi$.

Certainly, some other relationships like fixed-point identities are also valid in $\mathcal{LP}$-$\mathcal{CTL}^*$ (see [17]) but they will not be involved in this paper.

We can now bound our consideration with those $\mathcal{LP}$-$\mathcal{CTL}$ formulae which are constructed using only $\neg$, $\wedge$, $\mathbf{EX}_c$, $\mathbf{EY}_c$, $\mathbf{EG}_L$ and $\mathbf{EU}_L$. Let M be a transition system $TS(\Pi, \mathcal{A}^*) = (D, \mathcal{C}, d_{init}, \mathcal{T})$ of Π over $\mathcal{A}^*$. It should be noticed that M is, in general, infinite. Therefore, to obtain an effective model checking procedure we need a construction that will model the behaviour of M w.r.t. a target formula φ.

For every basic predicate $P \in \mathcal{P}$ let $A_P = (Q_P, \mathcal{A}, init_P, \delta_P, F_P)$ be a *minimal* DFA recognizing this language. Here Q_P is a finite set of states, $init_P$ is an initial state, F_P is a set of accepting states and $\delta_P : Q_P \times \mathcal{A} \to Q_P$ is a transition function. The latter can be extended to the set $\mathcal{A}^*$ in the usual fashion:

$$\delta_P(q_P, \varepsilon) = q_P \quad \textbf{and} \quad \delta_P(q_P, \gamma a) = \delta_P(\delta_P(q_P, \gamma), a).$$

Let $P_1, P_2, \ldots, P_k$ be all basic predicates occurred in the formula φ. Given a transducer $\Pi = (Q, \mathcal{C}, \mathcal{A}, q_{init}, T)$ and a formula φ, we build a *checking machine* — a transducer $\mathcal{M} = (\hat{Q}, \mathcal{C}, \mathcal{A}, \hat{q}_{init}, \hat{T})$, where

- $\hat{Q} = Q \times Q_{P_1} \times \ldots \times Q_{P_k}$ is a set of states (to avoid misunderstanding we will call them metastates);
- $\hat{q}_{init} = (q_{init}, init_{P_1}, \ldots, init_{P_k})$ is an initial metastate;
- $\hat{T} \subseteq \hat{Q} \times \mathcal{C} \times \hat{Q} \times \mathcal{A}^*$ is a transition relation, such that:

$$(\hat{\mathbf{q}}', \boldsymbol{c}, \hat{\boldsymbol{q}}'', \boldsymbol{h}) \in \hat{\boldsymbol{q}} \;\Leftrightarrow\; \begin{cases} (\boldsymbol{q}'_{\boldsymbol{\pi}}, \boldsymbol{c}, \boldsymbol{q}''_{\boldsymbol{\pi}}, \boldsymbol{h}) \in \boldsymbol{T} \textbf{ and} \\ \quad \boldsymbol{\delta}_{\boldsymbol{P_j}}(\boldsymbol{q}'_{\boldsymbol{P_j}}, \boldsymbol{h}) = \boldsymbol{q}''_{\boldsymbol{P_j}} \\ \qquad \textbf{for all } \boldsymbol{j}, \mathbf{1} \le \boldsymbol{j} \le \boldsymbol{k}. \end{cases}$$

Thus, every metastate is a tuple $\hat{q} = (q_0, q_1, \dots, q_k)$ such that $q_0 \in Q$ and $q_j \in Q_{P_j}$ for every j, $1 \le j \le k$, and the transition relation $\hat{T}$ synchronizes transitions of Π and the automata $A_{P_1}, \dots, A_{P_k}$ in response to every signal c. Recall that the elements of the free monoid are words s from $\mathcal{A}^*$. The checking machine $\mathcal{M}$ induces a binary relation $\sim$ on the set D: for an arbitrary pair $d' = (q', s')$ and $d'' = (q'', s'')$ of states of computation of Π over $\mathcal{A}^*$

$$\boldsymbol{d}' \sim \boldsymbol{d}'' \;\Leftrightarrow\; \begin{cases} \boldsymbol{q}' = \boldsymbol{q}'' \textbf{ and} \\ \boldsymbol{\delta}_{\boldsymbol{P_j}}(\boldsymbol{init}_{\boldsymbol{P_j}}, \boldsymbol{s}') = \boldsymbol{\delta}_{\boldsymbol{P_j}}(\boldsymbol{init}_{\boldsymbol{P_j}}, \boldsymbol{s}'') \textbf{ for all } \boldsymbol{j}. \end{cases}$$

The relation $\sim$ is clearly an equivalence relation of finite index, and every equivalence class of states of computation in $\boldsymbol{M}$ corresponds to a metastate of the checking machine $\boldsymbol{\mathcal{M}}$. As it can be seen from the definition of $\sim$, if two states of computation $\boldsymbol{d}'$ and $\boldsymbol{d}''$ are equivalent and there is a trajectory $\boldsymbol{tr}' = (\boldsymbol{d}', \boldsymbol{\alpha}')$ in $\boldsymbol{M}$, where $\boldsymbol{\alpha}' = (\boldsymbol{c_1}, \boldsymbol{d}'_{\boldsymbol{1}}), (\boldsymbol{c_2}, \boldsymbol{d}'_{\boldsymbol{2}}), \dots$, from one of these states, then there is also a corresponding trajectory $\boldsymbol{tr}'' = (\boldsymbol{d}'', \boldsymbol{\alpha}'')$, where $\boldsymbol{\alpha}'' = (\boldsymbol{c_1}, \boldsymbol{d}''_{\boldsymbol{1}}), (\boldsymbol{c_2}, \boldsymbol{d}''_{\boldsymbol{2}}), \dots$ from the other state, such that $\boldsymbol{d}'_{\boldsymbol{i}} \sim \boldsymbol{d}''_{\boldsymbol{i}}$ holds for every $\boldsymbol{i}, \boldsymbol{i} \ge \mathbf{1}$. Actually, this means that $\sim$ is a bisimulation relation on the state space of the transition system $\boldsymbol{M}$. It is well known (see [3, 8]) that bisimulation preserves the satisfiability of $\boldsymbol{CTL}$ formulae. The Proposition below shows that the same is true for $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}$. This means that the checking machine provides a finite contraction of the infinite transition system $\boldsymbol{M} = \boldsymbol{TS}(\boldsymbol{\Pi}, \boldsymbol{\mathcal{A}}^*)$ w.r.t. satisfiability of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}$ formulae.

Proposition 3. *Suppose that* $\boldsymbol{d}'$ *and* $\boldsymbol{d}''$ *are two states of computation in* $\boldsymbol{M}$ *such that* $\boldsymbol{d}' \sim \boldsymbol{d}''$. *Then* $\boldsymbol{M}, \boldsymbol{d}' \vDash \boldsymbol{\varphi} \;\Leftrightarrow\; \boldsymbol{M}, \boldsymbol{d}'' \vDash \boldsymbol{\varphi}$.

Proof. It is carried out by induction on the nesting depth of $\boldsymbol{\varphi}$. When $\boldsymbol{\varphi}$ is a basic predicate the assertion follows from the definition of $\sim$. The cases when $\boldsymbol{\varphi} = \neg\boldsymbol{\psi}$ and $\boldsymbol{\varphi} = \boldsymbol{\psi_1} \wedge \boldsymbol{\psi_2}$ are obvious. We focus only on the case of $\boldsymbol{\varphi} = \mathbf{E}[\boldsymbol{\psi}\, \mathbf{U}_{\boldsymbol{L}} \boldsymbol{\chi}]$; the other cases when $\boldsymbol{\varphi}$ is of the form $\mathbf{EX}_{\boldsymbol{c}}\boldsymbol{\psi}$, $\mathbf{EY}_{\boldsymbol{c}}\boldsymbol{\psi}$, or $\mathbf{EG}_{\boldsymbol{L}}\boldsymbol{\psi}$ can be treated similarly.
Suppose that $\boldsymbol{M}, \boldsymbol{d}' \vDash \mathbf{E}[\boldsymbol{\psi}\, \mathbf{U}_{\boldsymbol{L}} \boldsymbol{\chi}]$. Then, by the definition of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}$ semantics, there exists a trajectory $\boldsymbol{tr}' = (\boldsymbol{d}', \boldsymbol{\alpha}')$, such that $\boldsymbol{M}, \boldsymbol{tr}' \vDash \boldsymbol{\psi}\, \mathbf{U}_{\boldsymbol{L}} \boldsymbol{\chi}$ and $\boldsymbol{\alpha}' = (\boldsymbol{c_1}, \boldsymbol{d}'_{\boldsymbol{1}}), (\boldsymbol{c_2}, \boldsymbol{d}'_{\boldsymbol{2}}), \dots$. As it was noticed above, there is also a corresponding trajectory $\boldsymbol{tr}'' = (\boldsymbol{d}'', \boldsymbol{\alpha}'')$ in $\boldsymbol{M}$, where $\boldsymbol{\alpha}'' = (\boldsymbol{c_1}, \boldsymbol{d}''_{\boldsymbol{1}}), (\boldsymbol{c_2}, \boldsymbol{d}''_{\boldsymbol{2}}), \dots$, such that $\boldsymbol{d}'_{\boldsymbol{i}} \sim \boldsymbol{d}''_{\boldsymbol{i}}$ holds for every $\boldsymbol{i}, \boldsymbol{i} \ge \mathbf{1}$. Then, by induction hypotheses, $\boldsymbol{M}, \boldsymbol{d}'_{\boldsymbol{i}} \vDash \boldsymbol{\psi} \Leftrightarrow \boldsymbol{M}, \boldsymbol{d}''_{\boldsymbol{i}} \vDash \boldsymbol{\psi}$ and $\boldsymbol{M}, \boldsymbol{d}'_{\boldsymbol{i}} \vDash \boldsymbol{\chi} \Leftrightarrow \boldsymbol{M}, \boldsymbol{d}''_{\boldsymbol{i}} \vDash \boldsymbol{\chi}$ hold for every $\boldsymbol{i}, \boldsymbol{i} \ge \mathbf{1}$.
Since $\boldsymbol{M}, \boldsymbol{tr}' \vDash \boldsymbol{\psi}\, \mathbf{U}_{\boldsymbol{L}} \boldsymbol{\chi}$, there exists $\boldsymbol{i}$ such that

1. $c_1 c_2 \dots c_i \in L$ and $M, tr'|^i \vDash \chi$;
2. for all $j < i$ if $c_1 c_2 \dots c_j \in L$ then $M, tr'|^j \vDash \psi$.

However, taking into account the fact that ψ and χ are state formulas, we must recognize that $M, tr''|^i \vDash \chi$ and that $M, tr''|^j \vDash \psi$ every time when $M, tr'|^j \vDash \psi$. Thus, we arrive at the conclusion that $M, tr'' \vDash \psi \mathbf{U}_L \chi$ and, hence, $M, d'' \vDash \mathrm{E}[\psi \mathbf{U}_L \chi]$. ■

Each metastate $\hat{\boldsymbol{q}} = (\boldsymbol{q_0}, \boldsymbol{q_1}, \dots, \boldsymbol{q_k})$ of the checking machine $\boldsymbol{\mathcal{A}}$ represents an equivalence class $\boldsymbol{D_{\hat{q}}}$ which includes all states $\boldsymbol{d} = (\boldsymbol{q}, \boldsymbol{h}) \in \boldsymbol{D}$ such that $\boldsymbol{q} = \boldsymbol{q_0}$ and $\boldsymbol{\delta_{P_j}}(\boldsymbol{init_{P_j}}, \boldsymbol{h}) = \boldsymbol{q_j}$ for all $\boldsymbol{j}, \boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{k}$. By using Proposition 3, we can correctly introduce a new satisfiability relation $\vDash_{\boldsymbol{0}}$ on the metastates of the checking machine:

$$\hat{\boldsymbol{q}} \underset{\boldsymbol{0}}{\vDash} \boldsymbol{\varphi} \quad \Leftrightarrow \quad \textbf{for some } \boldsymbol{d} \in \boldsymbol{D_{\hat{q}}}\colon \ \boldsymbol{M}, \boldsymbol{d} \vDash \boldsymbol{\varphi}\ .$$

Not only the states of the transition system $\boldsymbol{M} = \boldsymbol{TS}(\boldsymbol{\Pi}, \boldsymbol{S})$ correspond to the metastates of the checking machine $\boldsymbol{\mathcal{M}}$, but also there is a relationship between the trajectories in $\boldsymbol{M}$ and the traces in $\boldsymbol{\mathcal{M}}$ (they can be quite naturally called metatrajectories). More formally, every trajectory $\boldsymbol{tr} = (\boldsymbol{d_0}, \boldsymbol{\alpha})$ in $\boldsymbol{M}$ with $\boldsymbol{\alpha} = (\boldsymbol{c_1}, \boldsymbol{d_1})(\boldsymbol{c_2}, \boldsymbol{d_2}) \dots$, corresponds to a *metatrajectory* $\widehat{\boldsymbol{tr}} = (\hat{\boldsymbol{q}}_{\boldsymbol{0}}, \hat{\boldsymbol{\alpha}})$, where $\hat{\boldsymbol{\alpha}} = (\boldsymbol{c_1}, \hat{\boldsymbol{q}}_{\boldsymbol{1}})(\boldsymbol{c_2}, \hat{\boldsymbol{q}}_{\boldsymbol{2}}) \dots$ is such that for all $\boldsymbol{i} \geq \boldsymbol{0}$: $\boldsymbol{d_i} \in \boldsymbol{D}_{\hat{\boldsymbol{q}}_i}$. It is easy to see that every metatrajectory $\widehat{\boldsymbol{tr}} = (\hat{\boldsymbol{q}}_{\boldsymbol{0}}, \hat{\boldsymbol{\alpha}})$ corresponds to the only trajectory $\boldsymbol{tr} = (\boldsymbol{d_0}, \boldsymbol{\alpha})$, which originates in a given state $\boldsymbol{d_0}$ from $\boldsymbol{D}_{\hat{\boldsymbol{q}}_0}$.

The well-known labeling algorithm for conventional $\boldsymbol{CTL}$ and ordinary Kripke structures can be now adapted in such a way as to cope with model checking problem for $\boldsymbol{\mathcal{LP}\text{-}CTL}$. The algorithm operates as follows. For every metastate $\hat{\boldsymbol{q}} \in \hat{\boldsymbol{Q}}$ of the checking machine $\boldsymbol{\mathcal{M}}$ it computes a set $\boldsymbol{label}(\hat{\boldsymbol{q}})$ of all subformulae of $\boldsymbol{\varphi}$ satisfied in $\hat{\boldsymbol{q}}$. More formally, let $\boldsymbol{Sub}(\boldsymbol{\varphi})$ be the minimal set of $\boldsymbol{\mathcal{LP}\text{-}CTL}$ formulae such that:

1. $\varphi \in Sub(\varphi)$;
2. if $\neg\psi \in Sub(\varphi)$ then $\psi \in Sub(\varphi)$;
3. if $\psi \wedge \chi \in Sub(\varphi)$ then $\psi,\ \chi \in Sub(\varphi)$;
4. if $\mathbf{EX}_c\psi \in Sub(\varphi)$, $\mathbf{EY}_c\psi \in Sub(\varphi)$ or $\mathbf{EG}_L\psi \in Sub(\varphi)$ then $\psi \in Sub(\varphi)$;
5. if $\mathbf{E}[\psi \mathbf{U}_L \chi] \in Sub(\varphi)$ then $\psi,\ \chi \in Sub(\varphi)$.

The algorithm builds incrementally the sets $label(\hat{\boldsymbol{q}})$ of all those $\psi \in Sub(\varphi)$ for which $\hat{\boldsymbol{q}} \vDash_0 \psi$ holds. At the first step every $label(\hat{\boldsymbol{q}})$ contains only basic predicates, i. e. $label(\hat{\boldsymbol{q}}) \subseteq Sub(\varphi) \cap \mathcal{P}$. Then, at $step\ i$ the algorithm processes those subformulae ψ whose nesting depth is $i - 1$. Every time when the algorithm adds a subformula ψ to $label(\hat{q})$ it thus detects that $\hat{q} \vDash_0 \psi$.

All we need now is to describe how the algorithm should process formulae of 7 types: basic predicate $\boldsymbol{P}$, $\boldsymbol{\neg\psi}$, $\boldsymbol{\psi_1 \wedge \psi_2}$, $\boldsymbol{EX_c\psi}$, $\boldsymbol{\mathbf{EY}_c\psi}$, $\boldsymbol{EG_L\psi}$ and $\boldsymbol{\mathbf{E}[\psi \mathbf{U}_L \chi]}$.

- A basic predicate P_i is added to $label(\hat{q})$ iff $\hat{q} = (q_0, q_1, \dots, q_i, \dots, q_k)$ and $\hat{q}_i \in F_{P_i},\ i \geq 1$;
- A subformula $\neg\psi$ is added to $label(\hat{q})$ iff $\psi \notin label(\hat{q})$;
- A subformula $\psi_1 \wedge \psi_2$ is added to $label(\hat{q})$ iff both $\psi_1, \psi_2 \in label(\hat{q})$;

- A subformula $\boldsymbol{EX}_c\psi$ is added to $label(\hat{q})$ iff there exists a transition $\hat{q} \overset{c,h}{\rightarrow} \hat{q}'$ such that $\psi \in label(\hat{q}')$;
- A subformula $\boldsymbol{EY}_c\psi$ is added to $label(\hat{q})$ iff there exists a transition $\hat{q} \overset{c,h}{\rightarrow} \hat{q}'$ such that $\psi \in label(\hat{q}')$ or a transition $\hat{q} \overset{c,h}{\rightarrow} \hat{q}'$ such that $c' \neq c$;
- To handle a subformula $\boldsymbol{E}[\psi\, \boldsymbol{U}_L \chi]$ we construct a directed labeled graph (DLG) $\Gamma_U(\mathcal{M}, L)$ as follows. Let $A_L = (Q_L, \mathcal{C}, init_L, \delta_L, F_L)$ be a minimal DFA that recognizes the language L. Then the nodes of $\Gamma_U(\mathcal{M}, L)$ are all pairs $(\hat{q}, q_L) \in \hat{Q} \times Q_L$. This DLG has an arc of the form $(\hat{q}', q_L') \overset{c,h}{\rightarrow} (\hat{q}'', q_L'')$ iff $\hat{q}' \overset{c,h}{\rightarrow} \hat{q}"$ is a transition of $\mathcal{M}$ and $\delta_L(q_L', c) = q_L''$.
  We then delete all those nodes $(\hat{q}, q_L)$ of $\Gamma_U(\mathcal{M}, L)$ for which the relations $\psi \notin label(\hat{q})$, $\chi \notin label(\hat{q})$ and $q_L \in F_L$ hold simultaneously and discard all arcs incoming to or outcoming from such nodes. A DLG thus reduced is denoted by $\Gamma'_U(\mathcal{M}, L)$.
  A subformula $E[\psi\, U_L \chi]$ is added to the set $label(\hat{q})$ iff $\Gamma'_U(\mathcal{M}, L)$ includes the node $(\hat{q}, init_L)$ and there exists a directed path in this graph from this node to some node $(\hat{q}', q_L')$ such that $\chi \in label(\hat{q}')$ and $q_L' \in F_L$.
- For a subformula $EG_L\psi$ we construct a DLG $\Gamma_G(\mathcal{M}, L)$ in the same fashion and delete all the nodes $(\hat{q}, q_L)$ for which the relations $\psi \notin label(\hat{q})$ and $q_L \in F_L$ hold simultaneously. As the result we obtain the reduced DLG $\Gamma'\_G\ (\mathcal{M}, L)$.
  The subformula $\mathbf{EG}_L\psi$ is added to the set $label(\hat{\boldsymbol{q}})$ iff $\Gamma'_G(\mathcal{M}, L)$ includes the node $(\hat{\boldsymbol{q}}, init_L)$ and there exists a directed path in this graph from this node to some nontrivial *strongly connected component* (SCC), that is, to a subgraph, every node of which is reachable from any other node by some non-empty path.

As soon as all the subformulae from $Sub(\varphi)$ (including the formula φ) are processed we obtain the result of the model checking as

$$\boldsymbol{\Pi} \vDash \boldsymbol{\varphi} \quad \Leftrightarrow \quad \boldsymbol{\varphi} \in \boldsymbol{label}(\hat{\boldsymbol{q}}_{\boldsymbol{init}}).$$

The correctness of this assertion is based on the following relationship: $\hat{\boldsymbol{q}} \vDash_{\mathbf{0}} \boldsymbol{\varphi} \Leftrightarrow \boldsymbol{\varphi} \in \boldsymbol{label}(\hat{\boldsymbol{q}})$. It can be proved by applying induction on the nesting depth of formulae with the help of Proposition 3. We also need Propositions 4 and 5 to justify the induction step for formulae of the form $\mathbf{E}[\boldsymbol{\psi}\, \mathbf{U}_L \boldsymbol{\chi}]$ and $\mathbf{EG}_L\boldsymbol{\psi}$.
Suppose, that for every metastate $\hat{\boldsymbol{q}} \in \hat{\boldsymbol{Q}}$ it is true that $\hat{\boldsymbol{q}} \vDash_{\mathbf{0}} \boldsymbol{\psi} \Leftrightarrow \boldsymbol{\psi} \in \boldsymbol{label}(\hat{\boldsymbol{q}})$ and $\hat{\boldsymbol{q}} \vDash_{\mathbf{0}} \boldsymbol{\chi} \Leftrightarrow \boldsymbol{\chi} \in \boldsymbol{label}(\hat{\boldsymbol{q}})$. This statement is used as an inductive hypothesis.

Proposition 4. *Let $\hat{\boldsymbol{q}}_{\mathbf{0}} \in \hat{\boldsymbol{Q}}$ be an arbitrary metastate in $\boldsymbol{\mathcal{M}}$. Then $\hat{\boldsymbol{q}}_{\mathbf{0}} \vDash_{\mathbf{0}} \boldsymbol{E}[\boldsymbol{\psi}\, \boldsymbol{U}_L \boldsymbol{\chi}]$ iff some node $(\hat{\boldsymbol{q}}', \boldsymbol{q}_L')$ in DLG $\boldsymbol{\Gamma}'_U(\boldsymbol{\mathcal{M}}, \boldsymbol{L})$, such that $\hat{\boldsymbol{q}}' \vDash_{\mathbf{0}} \boldsymbol{\chi}$ and $\boldsymbol{q}_L' \in \boldsymbol{F}_L$, is reachable from the node $(\hat{\boldsymbol{q}}_{\mathbf{0}}, \boldsymbol{init}_L)$ by a directed path.*

Proposition 5. *Let* $\widehat{\boldsymbol{q}}_{\boldsymbol{0}} \in \widehat{\boldsymbol{Q}}$ *be an arbitrary metastate in* $\boldsymbol{\mathcal{M}}$. *Then* $\widehat{\boldsymbol{q}} \vDash_{\boldsymbol{0}} \boldsymbol{EG_L\psi}$ *iff some nontrivial strongly connected component is reachable from the node* $(\widehat{\boldsymbol{q}}, \boldsymbol{init_L})$ *in DLG* $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$ *by a directed path.*

The proofs of these Propositions are straightforward adaptations of the correctness proof of the tabular model checking algorithm for $\boldsymbol{CTL}$ which is discussed in much details in [8]. However, for completeness of the exposition we give here a proof of Proposition 5. The proof of Proposition 4 follows the similar line of reasoning.

Proof of Proposition 5 (Sketch).
(⇒) Suppose, that $\widehat{\boldsymbol{q}}_{\boldsymbol{0}} \vDash_{\boldsymbol{0}} \mathbf{EG}_{\boldsymbol{L}}\boldsymbol{\psi}$. Consider an arbitrary state $\boldsymbol{d_0} \in \boldsymbol{D}_{\widehat{\boldsymbol{q}}_{\boldsymbol{0}}}$. Then, by definition of $\vDash_{\boldsymbol{0}}$ and by Proposition 3, it is true that $\boldsymbol{M}, \boldsymbol{d_0} \vDash \mathbf{EG}_{\boldsymbol{L}}\boldsymbol{\psi}$. This means that there is a trajectory $\boldsymbol{tr} = (\boldsymbol{d_0}, \boldsymbol{\alpha})$, where $\boldsymbol{\alpha} = (\boldsymbol{c_1}, \boldsymbol{d_1}), (\boldsymbol{c_2}, \boldsymbol{d_2}), ...$, such that $\boldsymbol{M}, \boldsymbol{tr} \vDash \mathbf{G}_{\boldsymbol{L}}\boldsymbol{\psi}$. By the semantics of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}^*$, $\boldsymbol{M}, \boldsymbol{d_i} \vDash \boldsymbol{\psi}$ holds for every $\boldsymbol{i}$ such that $\boldsymbol{c_1 c_2} \ldots \boldsymbol{c_i} \in \boldsymbol{L}$.
Consider now the corresponding metatrajectory $\widehat{\boldsymbol{tr}} = (\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \widehat{\boldsymbol{\alpha}})$ in the checking machine, where $\widehat{\boldsymbol{\alpha}} = (\boldsymbol{c_1}, \widehat{\boldsymbol{q}}_{\boldsymbol{1}}), (\boldsymbol{c_2}, \widehat{\boldsymbol{q}}_{\boldsymbol{2}}), ...$, and let

$$\boldsymbol{\pi} = (\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \boldsymbol{init_L}) \xrightarrow{\boldsymbol{c_1,h_1}} (\widehat{\boldsymbol{q}}_{\boldsymbol{1}}, \boldsymbol{q_{1L}}) \xrightarrow{\boldsymbol{c_2,h_2}} (\widehat{\boldsymbol{q}}_{\boldsymbol{2}}, \boldsymbol{q_{2L}}) \xrightarrow{\boldsymbol{c_3,h_3}} \ldots,$$

be the respective path in the DLG $\boldsymbol{\Gamma_G(\mathcal{M}, L)}$ which originates in the node $(\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \boldsymbol{init_L})$. Relying on Proposition 3 and taking into account the fact that $\boldsymbol{q_{iL}} = \boldsymbol{\delta_L(init_L, c_1 c_2 \ldots c_i)}$ for every $\boldsymbol{i}, \boldsymbol{i} \geq \mathbf{0}$, we may conclude that $\widehat{\boldsymbol{q}}_{\boldsymbol{i}} \vDash_{\boldsymbol{0}} \boldsymbol{\psi}$ holds for every $\boldsymbol{i}$ such that $\boldsymbol{q_{iL}} \in \boldsymbol{F}$. By induction hypothesis, $\widehat{\boldsymbol{q}}_{\boldsymbol{i}} \vDash_{\boldsymbol{0}} \boldsymbol{\psi}$ is equivalent to $\boldsymbol{\psi} \in \boldsymbol{label}(\widehat{\boldsymbol{q}}_{\boldsymbol{i}})$. Therefore, by definition of DLG $\boldsymbol{\Gamma_G(\mathcal{M}, L)}$ the path $\boldsymbol{\pi}$ is the infinite path which is entirely contained in the $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$. Due to the finiteness of $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$, this path may be represented as a concatenation $\boldsymbol{\pi} = \boldsymbol{\pi_1 \pi_2}$, where $\boldsymbol{\pi_1}$ is a finite path, and $\boldsymbol{\pi_2}$ is an infinite path passing through each of its nodes infinitely often. It is clear that the set $\boldsymbol{V(\pi_2)}$ of all nodes of $\boldsymbol{\pi_2}$ is included in some strongly connected component. Thus, a nontrivial strongly connected component is reachable from the node $(\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \boldsymbol{init_L})$ in DLG $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$.

(⇐) Suppose, that a nontrivial strongly connected component is reachable from the node $(\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \boldsymbol{init_L})$ in DLG $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$. Then there exists an infinite path

$$\boldsymbol{\pi} = (\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \boldsymbol{init_L}) \xrightarrow{\boldsymbol{c_1,h_1}} (\widehat{\boldsymbol{q}}_{\boldsymbol{1}}, \boldsymbol{q_{1L}}) \xrightarrow{\boldsymbol{c_2,h_2}} (\widehat{\boldsymbol{q}}_{\boldsymbol{2}}, \boldsymbol{q_{2L}}) \xrightarrow{\boldsymbol{c_3,h_3}} \ldots,$$

in $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$ from the node $(\widehat{\boldsymbol{q}}_{\boldsymbol{0}}, \boldsymbol{init_L})$ Consider now the sequence of the first components $\widehat{\boldsymbol{q}}_{\boldsymbol{i}}$ of all nodes $(\widehat{\boldsymbol{q}}_{\boldsymbol{i}}, \boldsymbol{q_{iL}}), \boldsymbol{i} \geq \mathbf{0}$, occurred in this path. By the definition of the DLG $\boldsymbol{\Gamma'_G(\mathcal{M}, L)}$,

1. this sequence is a metatrajectory $\widehat{tr}$ in the checking machine $\mathcal{M}$,
2. $\psi \in label(\hat{q}_i)$ holds for every node $(\hat{q}_i, q_{iL})$ such that $q_{iL} \in F_L$.

By the induction hypothesis, the latter implies $\hat{q}_i \vDash_0 \psi$ for every metastate $\widehat{\boldsymbol{q}}_{\boldsymbol{i}}$ in this trajectory such that $c_1 c_2 \ldots c_i \in L$. Consider an arbitrary state $d_0 \in D_{\hat{q}_0}$ and a

trajectory $tr = (d_0, \alpha)$ in M, where $\alpha = (c_1, d_1), (c_2, d_2), \ldots$, which corresponds to $\widehat{tr}$. By definition of $\models_0$ and Proposition 3, $M, d_i \models \psi$ holds for every i such that $c_1 c_2 \ldots c_i \in L$. Then, according to the semantics of $\mathcal{LP}$-$\mathcal{CTL}^*$, $M, tr \models \mathbf{G}_L \psi$, and, hence, $M, d_0 \models \mathbf{EG}_L \psi$. Thus, by referring once again to definition of $\models_0$, we arrive at the conclusion that $\widehat{\boldsymbol{q}}_0 \models_0 \mathbf{EG}_L \psi$. ■

Now we estimate the complexity of the model checking algorithm for $\mathcal{LP}$-$\mathcal{CTL}$ described above. By the *size of a transducer* $\boldsymbol{\Pi} = (\boldsymbol{Q}, \boldsymbol{\mathcal{C}}, \boldsymbol{\mathcal{A}}, \boldsymbol{q_{init}}, \boldsymbol{T})$ we will mean the sum $\| \boldsymbol{\Pi} \| = |\boldsymbol{Q}| + |\boldsymbol{T}|$. The *size of a formula* $\boldsymbol{\varphi}$ is defined as follows. Suppose that basic predicates $\{\boldsymbol{P_i}\}_{\boldsymbol{i=1}}^{\boldsymbol{k}}$ occurred in $\boldsymbol{\varphi}$ are recognized by minimal DFAs $\{\boldsymbol{A_{P_i}} = \boldsymbol{Q_{P_i}}, \boldsymbol{\mathcal{A}}, \boldsymbol{init_{P_i}}, \boldsymbol{\delta_{P_i}}, \boldsymbol{F_{P_i}})\}_{\boldsymbol{i=1}}^{\boldsymbol{k}}$. Suppose also that environment patterns $\{\boldsymbol{L_i}\}_{\boldsymbol{i=1}}^{\boldsymbol{s}}$ used in $\boldsymbol{\varphi}$ are recognized by minimal DFAs $\{\mathbf{A}_{\mathbf{L_i}} = (\mathbf{Q}_{\mathbf{L_i}}, \boldsymbol{\mathcal{A}}, \mathbf{init}_{\mathbf{L_i}}, \boldsymbol{\delta}_{\mathbf{L_i}}, \mathbf{F}_{\mathbf{L_i}})\}_{\boldsymbol{i=1}}^{\boldsymbol{s}}$. Then the size of $\boldsymbol{\varphi}$ is the sum $\| \boldsymbol{\varphi} \| = |\boldsymbol{Sub}(\boldsymbol{\varphi})| + \sum_{\boldsymbol{i=1}}^{\boldsymbol{k}} |\boldsymbol{Q_{P_i}}| + \sum_{\boldsymbol{i=1}}^{\boldsymbol{s}} |\boldsymbol{Q_{L_i}}|$.

As it can be seen from the description of our model checking algorithm, the size of auxiliary graphs $\boldsymbol{\Gamma'_U}(\boldsymbol{\mathcal{M}}, \boldsymbol{L})$ and $\boldsymbol{\Gamma'_G}(\boldsymbol{\mathcal{M}}, \boldsymbol{L})$ used in this algorithm does not exceed the value $\| \boldsymbol{\Pi} \| \cdot \left(\prod_{\boldsymbol{i=0}}^{\boldsymbol{k}} |\boldsymbol{Q_{P_i}}| \right) \cdot \mathbf{max}(|\boldsymbol{Q_{L_i}}| : \mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{s})$. These graphs are processed in no more than $|\boldsymbol{Sub}(\boldsymbol{\varphi})|$ steps. So, the total time complexity of our model checking algorithm does not exceed the value $\| \boldsymbol{\Pi} \| \cdot |\boldsymbol{Sub}(\boldsymbol{\varphi})| \left(\prod_{\boldsymbol{i=0}}^{\boldsymbol{k}} |\boldsymbol{Q_{P_i}}| \right) \cdot \mathbf{max}(|\boldsymbol{Q_{L_i}}| : \mathbf{1} \leq \boldsymbol{i} \leq \boldsymbol{s})$ which is $\boldsymbol{O}(\| \boldsymbol{\Pi} \| \cdot \mathbf{2}^{\| \boldsymbol{\varphi} \|})$.

Because of these considerations, we get the following

Theorem 1. *Model checking of a finite state transducer* $\boldsymbol{\Pi}$ *operating over a free monoid against a formula* $\boldsymbol{\varphi} \in \mathcal{LP}$-$\mathcal{CTL}$ *can be performed in time* $\boldsymbol{O}(\| \boldsymbol{\Pi} \| \cdot \mathbf{2}^{\| \boldsymbol{\varphi} \|})$.

When a more general case of model checking problem of FSTs against $\mathcal{LP}$-$\mathcal{CTL}^*$ formulae is concerned we can rely on the well-known combining approach which is based on the interleaving application of model checking algorithms for $\mathcal{CTL}$ and $\mathcal{LTL}$. The details can be found in [8]. The similar procedure for $\mathcal{LP}$-$\mathcal{CTL}^*$ can be obtained in the same fashion by means of $\mathcal{LP}$-$\mathcal{CTL}$ model checking algorithm described above and $\mathcal{LP}$-$\mathcal{LTL}$ model checking algorithm developed in . Since this approach does not take into account any specific features of $\mathcal{LP}$-$\mathcal{CTL}^*$ formulae, we will not give a complete description of it.

5. $\mathcal{LP}$-$\mathcal{LTL}^*$ and ordinary Kripke structures

In this section, we consider the model checking problem for two subfamilies of $\mathcal{LP}$-$\mathcal{CTL}^*$ whose semantics can be defined on ordinary Kripke structures.

Recall, that a *Kripke structure* over a finite set $\boldsymbol{AP}$ of atomic propositions is a quadruple $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{q_{init}}, \boldsymbol{R}, \boldsymbol{\rho})$, where $\boldsymbol{Q}$ is a finite set of states which includes an initial state $\boldsymbol{q_{init}}$, $\boldsymbol{R} \subseteq \boldsymbol{Q} \times \boldsymbol{Q}$ is a transition relation and $\boldsymbol{\rho}: \boldsymbol{Q} \to \boldsymbol{2^{AP}}$ is a *labeling function* which for each state $\boldsymbol{q}$ gives a matching set $\boldsymbol{\rho(q)} \subseteq \boldsymbol{AP}$ of all atomic propositions that are evaluated to $\boldsymbol{true}$ in this state. As usual, the *size* of $\boldsymbol{M}$ is the sum $\| \boldsymbol{M} \| = |\boldsymbol{Q}| + |\boldsymbol{R}|$. Below we present two modifications of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}^*$ that are well suited for model checking of Kripke structures.

Given a Kripke structure $\boldsymbol{M} = (\boldsymbol{Q}, \boldsymbol{q_{init}}, \boldsymbol{R}, \boldsymbol{L})$, consider a set of $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{CTL}^*$ formulae where $\boldsymbol{\mathcal{L}}$is a family of regular languages over one-letter alphabet $\{\boldsymbol{c}\}$ and $\boldsymbol{\mathcal{P}} = \boldsymbol{AP}$ (we denote this formulae by $\boldsymbol{\mathcal{LP}}$-1-$\boldsymbol{CTL}^*$) and a transition system $\boldsymbol{M_c} = (\boldsymbol{Q}, \{\boldsymbol{c}\}, \boldsymbol{q_{init}}, \boldsymbol{R_c}, \boldsymbol{L})$ where $(\boldsymbol{q'}, \boldsymbol{c}, \boldsymbol{q''}) \in \boldsymbol{R_c}$ iff $(\boldsymbol{q'}, \boldsymbol{q''}) \in \boldsymbol{R}$. Then for $\boldsymbol{q} \in \boldsymbol{Q}$ the relation $\boldsymbol{q} \vDash \boldsymbol{P}$ holds iff $\boldsymbol{P} \in \boldsymbol{\rho(q)}$. The semantics of more complex formulae is defined exactly as in Section 3.

Some $\boldsymbol{\mathcal{LP}}$-1-$\boldsymbol{CTL}^*$ formulae have an ability to keep track of the number of steps of the run. For example, an $\boldsymbol{\mathcal{LP}}$-1-$\boldsymbol{LTL}$ formula $\mathbf{AG}_{\boldsymbol{L}}\, \boldsymbol{\varphi}$, where $\boldsymbol{L} = \{\boldsymbol{c^{2n}}\}$ is a regular language which contains all 1-letter words of even length, expresses the assertion that $\boldsymbol{\varphi}$ holds at every even step of a run. By using the techniques of Ehrenfeucht-Fraisse games for Temporal Logics developed and studied in [11] one can prove that this property cannot be specified by means of usual $\boldsymbol{LTL}$. This certifies that $\boldsymbol{\mathcal{LP}}$-1-$\boldsymbol{CTL}^*$ is more expressive than $\boldsymbol{CTL}^*$ and justifies its use as a new specification language for finite state transducers and Kripke structures.

Observe, that given a set $\boldsymbol{AP}$ of all atomic propositions used in formulae we can use the $\boldsymbol{M_c}$ directly as a checking machine $\boldsymbol{\mathcal{M}}$ for the algorithm described in Section 4. Suppose that formula $\boldsymbol{\varphi}$ refers to 1-letter regular languages $\boldsymbol{L_1}, \boldsymbol{L_2}, \dots, \boldsymbol{L_s}$ as the parameters of temporal operators, and every language $\boldsymbol{L_i}$, $\boldsymbol{1 \le i \le s}$, is recognized by a DFA with a set of states $\boldsymbol{Q_{L_i}}$. Then the size of the graphs used in this algorithm does not exceed the value $\| \boldsymbol{M} \| \cdot \mathbf{max}(|\boldsymbol{Q_{L_i}}| : \boldsymbol{1 \le i \le s})$ which is $\boldsymbol{O}(\| \boldsymbol{M} \| \cdot \| \boldsymbol{\varphi} \|)$, where $\| \boldsymbol{\varphi} \| = |\boldsymbol{Sub}(\boldsymbol{\varphi})| + \sum_{\boldsymbol{i=0}}^{\boldsymbol{s}} |\boldsymbol{Q_{L_i}}|$.

Another modification of the Kripke structure $\boldsymbol{M}$ allows one to encode more detailed information of the computation flow. Let $\boldsymbol{\Sigma} = \boldsymbol{2^{AP}}$. For each state $\boldsymbol{q}$ in $\boldsymbol{M}$ there exists a letter $\boldsymbol{\sigma_{\rho(q)}} \in \boldsymbol{\Sigma}$ corresponding to the label $\boldsymbol{\rho(q)}$ assigned to this state.

Let $\boldsymbol{M_{AP}} = (\boldsymbol{Q} \cup \{\boldsymbol{err}\}, \boldsymbol{q_{init}}, \boldsymbol{R_{AP}}, \boldsymbol{\rho_{AP}})$ be a transition system for $\boldsymbol{M}$, where for every $\boldsymbol{q} \in \boldsymbol{Q}$ the following equalities hold: $\boldsymbol{\rho_{AP}(q)} = \boldsymbol{\rho(q)}$, $\boldsymbol{\rho_{AP}(err)} = \{\boldsymbol{err}\}$ and $\boldsymbol{R_{AP}} \subseteq \boldsymbol{Q} \times \boldsymbol{2^{AP}} \times \boldsymbol{Q}$ is a minimal transition relation such that:

- for each transition (q', q'') of the Kripke structure M there exists a *fair transition* $(q', \sigma_{\rho(q'')}, q'')$ and *erroneous transitions* (q', σ, err) for each $\sigma \neq \sigma_{\rho(q'')}$;
- $(err, \sigma, err) \in R_{AP}$ holds for each $\sigma \in \Sigma$ and $(err, \sigma, q) \notin R_{AP}$ holds for each $q \neq err$.

Then consider a specification language $\boldsymbol{\mathcal{LP}}$-$\boldsymbol{n}$-$\boldsymbol{CTL}^*$ which is a set of all such formulae where $\mathcal{L}$ is a family of regular languages over Σ and $\mathcal{P}$ = AP. To model

check a transition system M_{AP} against these formulae one needs to process only the states in Q and only the fair transitions. To do so, we replace all state formulae of type $\mathbf{A}\varphi$ with $\boldsymbol{A}(\boldsymbol{G}\neg err \longrightarrow \varphi)$ and all state formulae of type $\boldsymbol{E}\varphi$ with $\boldsymbol{E}(\boldsymbol{G}\neg err \wedge \varphi)$. The transition system M_{AP} thus obtained may as well be used as a checking machine for the model checking algorithm described in Section 4.
Thereby, the following theorem holds.

Theorem 2.

1. *There exists an algorithm for model checking of a Kripke structure M against a formula $\varphi \in \boldsymbol{\mathcal{LP}\text{-}1\text{-}CTL}$ with time complexity $O(\| M \| \cdot \| \varphi \|^2)$.*
2. *There exists an algorithm for model checking of a Kripke structure M against a formula $\varphi \in \boldsymbol{\mathcal{LP}\text{-}n\text{-}CTL}$ with time complexity $O\left(\| M \| \cdot \| \varphi \|^2 \cdot 2^{|AP|}\right)$.*

As it can be seen from this theorem, the exponential complexity of model checking procedure described in Section 4 is due to the language-theoretic nature of basic predicates used in $\boldsymbol{\mathcal{LP}\text{-}CTL^*}$.

6. Related papers and conclusion

Actually, the idea of providing parameterization of temporal operators is not new. In [27] right-linear grammar patterns were offered to define new temporal operators. The same kind of temporal patterns but specified by means of finite state automata were introduced in [18, 24]. For these extensions it was proved that they have the same expressiveness as $\boldsymbol{S1S}$ and that satisfiability checking problem in these logics is PSPACE-complete. We did not pursue a goal of merely expanding the expressive possibilities of $\boldsymbol{CTL^*}$; our aim was to make $\boldsymbol{CTL^*}$ more adequate for describing the behaviour of reactive systems. Almost the same kind of parametrization is used in Dynamic $\boldsymbol{LTL}$. However, our extension of $\boldsymbol{CTL^*}$ differs from that which was developed in [14], since in our logic basic predicates are also parameterized.
The $\boldsymbol{\mathcal{LP}\text{-}CTL^*}$ formulae allows one to specify and verify the behaviour of finite state transducers that operate over semigroups as well as classical Kripke structures. Moreover, when Kripke structures are concerned $\boldsymbol{\mathcal{LP}\text{-}CTL^*}$ has more expressive power than conventional temporal logics. But the place of $\boldsymbol{\mathcal{LP}\text{-}CTL^*}$ in the expressive hierarchy of specification languages, such as $\boldsymbol{S1S}$, $\boldsymbol{PDL}$ or $\boldsymbol{\mu}$-calculus, has not yet been established and remains a matter for our further research.
The results of this paper combined with the results of [17] provide positive solution to model checking for transducers over free semigroups. Free semigroups is the most simple algebraic structure which can be used for interpretation of basic actions performed by transducers when they are regarded as formal models of sequential reactive systems. Next, we are going to find out whether model checking algorithms could be built for transducers operating over more specific semigroups. Some preliminary results showed that this is not an easy problem. In [12] we proved that it is undecidable for the case of Abelian groups and free commutative semigroups.

It is also interesting how much the complexity of model checking algorithms for $\mathcal{LP}$-CTL^* depends on languages that are used as parameters of temporal operators. We assume that model checking problem becomes undecidable when context-free languages are allowed for this purpose. The complexity issues of model checking for regular variant of $\mathcal{LP}$-CTL^* also need further research. We assume that even for regular $\mathcal{LP}$-CTL this problem is PSPACE-complete.

As for practical application of the results obtained, the most important issue is that of adapting the existing means of working with finite automata to widely known model checking tools (like SPIN, ν-SMV, etc.) in order to be able to effectively implement the proposed model checking algorithms for $\mathcal{LP}$-CTL^*.

Acknowledgments

The authors of the article thank the anonymous reviewers for their valuable comments and advice on improving the article. This work was supported by the Russian Foundation for Basic Research, Grant N 18-01-00854.

References

[1]. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs. In Proceedings of 38-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2011, pp. 599-610

[2]. Alur R., Moarref S., and Topcu U. Pattern-based refinement of assume-guarantee specifications in reactive synthesis. In Proceedings of 21-st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2015, pp. 501-516.

[3]. Baier C., Katoen J. Principles of Model Checking, 2008, MIT Press.

[4]. Blattner M., Head T. The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences, 1979, vol. 1, pp. 45-49.

[5]. Blattner M., T. Head T. Single-valued a-transducers. Journal of Computer and System Sciences, vol. 15, 1977, pp. 310-327.

[6]. Culik K, Karhumaki J. The equivalence of finite-valued transducers (on HDTOL languages) is decidable. Theoretical Computer Science, 1986, vol. 47, pp. 71-84.

[7]. Bouajjani A., Jonsson B., Nilsson M., Touili T. Regular Model Checking. Proceedings of 12-th International Conference on Computer Aided Verification, 2000, p. 403-418.

[8]. Clarke (Jr.) E. M., Grumberg O., Peled D. A. Model Checking. MIT Press, 1999.

[9]. Diekert V., Rozenberg G. eds. The Book of Traces, 1995, World Scientific, Singapore.

[10]. Emerson E.A., Halpern J.Y. Decision procedures and expressiveness in the temporal logic of branching time. Journal of Computer and System Sciences, vol. 30, N 1, 1985, pp. 1–24.

[11]. Etessami K., WilkeT. An Until Hierarchy and Other Applications of and Ehrenfeucht-Fraisse Game for Temporal Logic. Information and Computation, vol. 160, 2000, pp. 88-108.

[12]. Gnatenko A.R., Zakharov V. A. On the complexity of verification of finite state machines over commutative semigroups. In Proceedings of the 18-th International Conference "Problems of Theoretical Cybernetics", 2017, pp. 68-70 (in Russian).

[13]. Griffiths T. The unsolvability of the equivalence problem for free nondeterministic generalized machines. Journal of the ACM, vol. 15, 1968, pp. 409-413.
[14]. HenriksenJ. G., Thiagarajan P.S. Dynamic linear time temporal logic. Annals of Pure and Applied Logic, vol. 96, 1999, pp.187-207.
[15]. Hu Q., D'Antoni L. Automatic Program Inversion using Symbolic Transducers. In Proceedings of the 38-th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017, pp. 376-389.
[16]. Ibarra O. The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. SIAM Journal on Computing, vol. 4, 1978, pp. 524-532.
[17]. Kozlova D. G., Zakharov V. A. On the model checking of sequential reactive systems. Proceedings of the25-th International Workshop on Concurrency, Specification and Programming (CS&P 2016), CEUR Workshop Proceedings, vol. 1698, 2016, pp. 233-244.
[18]. Kupferman O., Piterman N., Vardi M.Y. Extended Temporal Logic Revisited. In Proceedings of 12-th International Conference on Concurrency Theory, 2001, pp. 519-535.
[19]. Schutzenberger M. P. Sur les relations rationnelles. In Proceedings of Conference on Automata Theory and Formal Languages, 1975, pp. 209-213.
[20]. Sakarovitch J., de Souza R. On the decomposition of k-valued rational relations. In Proceedings of 25-th International Symposium on Theoretical Aspects of Computer Science, 2008, pp. 621-632.
[21]. Sakarovitch J., de Souza R. On the decidability of bounded valuedness for transducers. In Proceedings of the 33-rd International Symposium on Mathematical Foundations of Computer Science, 2008, pp. 588-600.
[22]. De Souza R. On the decidability of the equivalence for k-valued transducers. In Proceedings of 12-th International Conference on Developments in Language Theory, 2008, pp. 252-263.
[23]. Thakkar J., Kanade A., Alur R. A transducer-based algorithmic verification of retransmission protocols over noisy channels. In Proceedings of IFIP Joint International Conference on Formal Techniques for Distributed Systems, 2013, pp. 209-224.
[24]. Vardi M.Y., Wolper P. Yet Another Process Logic. Logic of Programs, 1983, pp. 501-512.
[25]. Veanes M., Hooimeijer P., Livshits B. et al. Symbolic finite state transducers: algorithms and applications. In Proceedings of the 39-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM SIGPLAN Notices, vol. 147, 2012, pp. 137-150.
[26]. Weber A. Decomposing finite-valued transducers and deciding their equivalence. SIAM Journal on Computing. vol. 22, 1993, pp. 175-202.
[27]. Wolper P. Temporal Logic Can Be More Expressive. Information and Control, vol. 56, N 1/2, 1983, pp. 72-99.
[28]. Wolper P., Boigelot B. Verifying systems with infinite but regular state spaces. In Proceedings of the 10-th Int. Conf. on Computer Aided Verification (CAV-1998), 1998, pp. 88-97.
[29]. Zakharov V.A. Equivalence checking problem for finite state transducers over semigroups. In Proceedings of the~6-th International Conference on Algebraic Informatics (CAI-2015), 2015, pp. 208-221.

О верификации конечных автоматов-преобразователей над полугруппами

[1] *А.Р. Гнатенко<gnatenko.cmc@gmail.com>*
[2] *В.А. Захаров<zakh@cs.msu.su>*
[1] *Московский государственный университет им. М. В. Ломоносова, 119991, Российская Федерация, Москва, Ленинские горы, д. 1*
[2] *Национальный исследовательский университет Высшая школа экономики, 101000, Российская Федерация, Москва, ул. Мясницкая, д. 20*

Аннотация. Последовательные реагирующие системы – это программы, которые взаимодействуют с окружением, получая от него сигналы или запросы, и реагируют на эти запросы, проводя операции с данными. Подобные системы могут служить моделью для многих программ: драйверов, систем реального времени, сетевых протоколов и др. В статье исследуются задача верификации программ такого вида. В качестве формальных моделей для реагирующих систем мы используем конечные автоматы-преобразователи, работающие над полугруппами. Для описания поведения автоматов-преобразователей введён новый язык спецификаций LP-CTL*. В его основу положена темпоральная логика CTL*. Этот язык спецификаций имеет две характерные особенности: 1) каждый темпоральный оператор снабжён регулярным выражением над входным алфавитом автомата, и 2) каждое атомарное высказывание задается регулярным выражением над выходным алфавитом автомата-преобразователя. В данной работе представлен табличный алгоритм проверки выполнимости формул LP-CTL* на моделях конечных автоматов-преобразователей, работающих над свободными полугруппами. Доказана корректность предложенного алгоритма и получена оценка его сложности. Кроме того, рассмотрен специальный фрагмент языка LP-CTL*, содержащий в качестве параметров темпоральных операторов только регулярные выражения над однобуквенным алфавитом. Показано, что этот фрагмента применим для спецификаций обычных моделей Крипке, и при этом его выразительные возможности превосходят обычную логику CTL*.

Ключевые слова: реагирующая система, автомат-преобразователь, верификация, проверка на модели, темпоральная логика, конечный автомат, регулярный язык.

DOI: 10.15514/ISPRAS-2018-30(3)-21

Для цитирования: Гнатенко А.Р., Захаров В.А. О верификации конечных автоматов-преобразователей над полугруппами. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 303-324 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-21

Список литературы

[1]. Alur R., Cerny P. Streaming transducers for algorithmic verification of single-pass list-processing programs. In Proceedings of 38-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, 2011, pp. 599-610

[2]. Alur R., Moarref S., and Topcu U. Pattern-based refinement of assume-guarantee specifications in reactive synthesis. In Proceedings of 21-st International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2015, pp. 501-516.
[3]. Baier C., Katoen J. Principles of Model Checking, 2008, MIT Press.
[4]. Blattner M., Head T. The decidability of equivalence for deterministic finite transducers. Journal of Computer and System Sciences, 1979, vol. 1, pp. 45-49.
[5]. Blattner M., T. Head T. Single-valued a-transducers. Journal of Computer and System Sciences, vol. 15, 1977, pp. 310-327.
[6]. Culik K, Karhumaki J. The equivalence of finite-valued transducers (on HDTOL languages) is decidable. Theoretical Computer Science, 1986, vol. 47, pp. 71-84.
[7]. Bouajjani A., Jonsson B., Nilsson M., Touili T. Regular Model Checking. Proceedings of 12-th International Conference on Computer Aided Verification, 2000, p. 403-418.
[8]. Clarke (Jr.) E. M., Grumberg O., Peled D. A. Model Checking. MIT Press, 1999.
[9]. Diekert V., Rozenberg G. eds. The Book of Traces, 1995, World Scientific, Singapore.
[10]. Emerson E.A., Halpern J.Y. Decision procedures and expressiveness in the temporal logic of branching time. Journal of Computer and System Sciences, vol. 30, N 1, 1985, pp. 1–24.
[11]. Etessami K., WilkeT. An Until Hierarchy and Other Applications of and Ehrenfeucht-Fraisse Game for Temporal Logic. Information and Computation, vol. 160, 2000, pp. 88-108.
[12]. Гнатенко А.Р., Захаров В.А. О сложности верификации конечных автоматов-преобразователей над коммутативными полугруппами. Материалы XVIII международной конференции «Проблемы теоретической кибернетики»' (Пенза, 20-24 июня, 2017), стр. 68-70.
[13]. Griffiths T. The unsolvability of the equivalence problem for free nondeterministic generalized machines. Journal of the ACM, vol. 15, 1968, pp. 409-413.
[14]. HenriksenJ. G., Thiagarajan P.S. Dynamic linear time temporal logic. Annals of Pure and Applied Logic, vol. 96, 1999, pp.187-207.
[15]. Hu Q., D'Antoni L. Automatic Program Inversion using Symbolic Transducers. In Proceedings of the 38-th ACM SIGPLAN Conference on Programming Language Design and Implementation, 2017, pp. 376-389.
[16]. Ibarra O. The unsolvability of the equivalence problem for Efree NGSM's with unary input (output) alphabet and applications. SIAM Journal on Computing, vol. 4, 1978, pp. 524-532.
[17]. Kozlova D. G., Zakharov V. A. On the model checking of sequential reactive systems. Proceedings of the25-th International Workshop on Concurrency, Specification and Programming (CS&P 2016), CEUR Workshop Proceedings, vol. 1698, 2016, pp. 233-244.
[18]. Kupferman O., Piterman N., Vardi M.Y. Extended Temporal Logic Revisited. In Proceedings of 12-th International Conference on Concurrency Theory, 2001, pp. 519-535.
[19]. Schutzenberger M. P. Sur les relations rationnelles. In Proceedings of Conference on Automata Theory and Formal Languages, 1975, pp. 209-213.
[20]. Sakarovitch J., de Souza R. On the decomposition of k-valued rational relations. In Proceedings of 25-th International Symposium on Theoretical Aspects of Computer Science, 2008, pp. 621-632.

[21]. Sakarovitch J., de Souza R. On the decidability of bounded valuedness for transducers. In Proceedings of the 33-rd International Symposium on Mathematical Foundations of Computer Science, 2008, pp. 588-600.

[22]. De Souza R. On the decidability of the equivalence for k-valued transducers. In Proceedings of 12-th International Conference on Developments in Language Theory, 2008, pp. 252-263.

[23]. Thakkar J., Kanade A., Alur R. A transducer-based algorithmic verification of retransmission protocols over noisy channels. In Proceedings of IFIP Joint International Conference on Formal Techniques for Distributed Systems, 2013, pp. 209-224.

[24]. Vardi M.Y., Wolper P. Yet Another Process Logic. Logic of Programs, 1983, pp. 501-512.

[25]. Veanes M., Hooimeijer P., Livshits B. et al. Symbolic finite state transducers: algorithms and applications. In Proceedings of the 39-th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM SIGPLAN Notices, vol. 147, 2012, pp. 137-150.

[26]. Weber A. Decomposing finite-valued transducers and deciding their equivalence. SIAM Journal on Computing. vol. 22, 1993, pp. 175-202.

[27]. Wolper P. Temporal Logic Can Be More Expressive. Information and Control, vol. 56, N 1/2, 1983, pp. 72-99.

[28]. Wolper P., Boigelot B. Verifying systems with infinite but regular state spaces. In Proceedings of the 10-th Int. Conf. on Computer Aided Verification (CAV-1998), 1998, pp. 88-97.

[29]. Zakharov V.A. Equivalence checking problem for finite state transducers over semigroups. In Proceedings of the~6-th International Conference on Algebraic Informatics (CAI-2015), 2015, pp. 208-221.

On the verification of strictly deterministic behavior of Timed Finite State Machines

E.M. Vinarskii <vinevg2015@gmail.com>
V.A. Zakharov <zakh@cs.msu.su>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. Finite State Machines (FSMs) are widely used as formal models for solving numerous tasks in software engineering, VLSI design, development of telecommunication systems, etc. To describe the behavior of a real-time system one could supply FSM model with clocks — a continuous time parameters with real values. In a Timed FSM (TFSM) inputs and outputs have timestamps, and each transition is equipped with a timed guard and an output delay to indicate time interval when the transition is active and how much time does it take to produce an output. A variety of algorithms for equivalence checking, minimization and test generation were developed for TFSMs in many papers. A distinguishing feature of TFSMs studied in these papers is that the order in which output letters occur in an output timed word does not depend on their timestamps. We think that such behavior of a TFSM is not realistic from the point of view of an outside observer. In this paper we consider a more advanced and adequate TFSM functioning; in our model the order in which outputs become visible to an outsider is determined not only by the order of inputs, but also by de lays required for their processing. When the same sequence of transitions is performed by a TFSM modified in a such way, the same outputs may follow in different order depending on the time when corresponding inputs become available to the machine. A TFSM is called strictly deterministic if every input timed word activates no more than one sequence of transitions (trace) and for any input timed word which activates this trace the letters in the output words always follows in the same order (but, maybe, with different timestamps). We studied the problem of checking whether a behavior of an improved model of TFSM is strictly deterministic. To this end we showed how to verify whether an arbitrary given trace in a TFSM is steady, i.e. preserves the same order of output letters for every input timed word which activates this trace. Further, having the criterion of trace steadiness, we developed an exhaustive algorithm for checking the property of strict determinacy of TFSMs. Exhaustive search in this case can hardly be avoided: we proved that determinacy checking problem for our model of TFSM is co-NP-hard.

Keywords: Timed Finite State Machines; strictly deterministic behavior

DOI: 10.15514/ISPRAS-2018-30(3)-22

For citation: Vinarskii E.M., Zakharov V.A. On the verification of strictly deterministic behaviour of Timed Finite State Machines. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 325-340. DOI: 10.15514/ISPRAS-2018-30(3)-22

1. Introduction

Finite State Machines (FSMs) are widely used as formal models for analysis and synthesis of information processing systems in software engineering, VLSI design, telecommunication, etc. The most attractive feature of this model of computation is its simplicity — many important synthesis and analysis problems (equivalence checking, minimization, test derivation, etc.) for classical FSMs can be solved in time which is almost linear or quadratic of the size of an FSM under consideration.

The concept of FSM is rather flexible. Since in many applications time aspects such as durations, delays, timeouts are very important, FSMs can be augmented with some additional features to describe the dependence of the behavior of a system on events occurring in real time. One of the most advanced timed extension of FSMs is the concept of Timed Automata which was developed and studied in [1]. Timed Automata are supplied with clocks (timers) for indicating real time moments, measuring durations of events, providing timeout effects. Transitions in such automata depends not only on the incoming of the outside messages and signals but also on the values of clocks. Further research showed that this model of computation is very expressive and captures many important features of real-time systems behavior. On the other side, Timed Automata in the full scope of their computing power are very hard for analysis and transformations. The reachability problem for Timed Automata is decidable [2], and, therefore, this model of computation is suitable for formal verification of real-time computer systems. But many other problems such as universality, inclusion, determinability, etc. are undecidable (see [2], [8]), and this hampers considerably formal analysis of Timed Automata.

When a Timed Automaton is capable to selectively reset timers, it can display rather sophisticated behavior which is very difficult for understanding and analysis. In some cases, such ability is very important; see, e.g. [9]. But a great deal of real-time programs and devices operate with timers much more simply: as soon as such a device switches to a new mode of operation (new state), it resets all timers. Timed Finite State Machines (TFSM) of this kind were studied in [5], [10], [13], [14]. TFSM has the only timer which it resets "automatically" as soon as it moves from one state to another. On the other hand, TFSMs, in contrast to Timed Automata introduced in [1], operate like transducers: they receive a sequence of input signals augmented with their timestamps (input timed word) and output a sequence of responses also labeled by timestamps (output timed word). The timestamps are real numbers which indicate the time when an input signal becomes available to a TFSM or an output response is generated. Transitions of a TFSM are equipped with time guards to indicate time intervals when transitions are active. Therefore, a reaction of a TFSM to an input signal depends not only on the signal but also on its timestamp. Some algorithms for equivalence checking, minimization and test generation were developed for TFSMs in [6], [5], [13], [14], [15]. It can be recognized that this model of TFSM combines a sufficient expressive power for modeling a wide class of real-time information processing systems and a developed algorithmic support.

As it was noticed above a behavior of a TFSM is characterized by a pair sequences: an input timed word and a corresponding output timed word. A distinguishing feature of TFSMs studied in [5], [10], [13], [14], [15] is that an output timed word is formed of timestamped output letters that follows in the same order as the corresponding input letters regardless of their timestamps. Meanwhile, suppose that a user of some file management system gives a command «Save» and immediately after that a command «Exit». Then if a file to be saved is small then the user will observe first a response «File is saved» and then a notification «File Management System is closed». But if a file has a considerable size then it takes a lot of time to close it. Therefore, it can happen that a user will detect first a notification «File Management System is closed» and then, some time later, he/she will be surprised to find an announcement «File is saved». Of course, the user may regard such behavior of the system enigmatic. But much worse if the order in which these notifications appear may vary in different sessions of the system. If a File Management System interacts with other service programs such an interaction will almost certainly lead to errors. However, if a behavior of TFSMs is defined as in the papers referred above then such a model can not adequately capture behavioral defects of real-time systems, similar to the one that was considered in the example.

To avoid this shortcoming of conventional TFSMs and to make their behavior more "realistic" from the point of view of an outside observer we offer some technical change to this model. We will assume that an output timed word consists of timestamped letters, and these letters always follow in ascending order of their timestamps regardless of an order in which the corresponding input letters entered a TFSM. In this model it may happen so that an input $\boldsymbol{b}$ follows an input $\boldsymbol{a}$ but a response to $\boldsymbol{b}$ appears before a response to $\boldsymbol{a}$ is computed. Clearly, the defect with File Management System discussed above becomes visible to an outside observer "through" the model of TFSMs thus modified.

At first sight, it may seem that this change only slightly complicates the analysis of the behavior of such models. But this is a false impression. In the initial model of TFSM the formation of an output timed word is carried out by local means for each state of the system. In our model this is a global task since to find the proper position of a timestamped output letter one should consider the run of TFSM as a whole. Therefore, even the problem of checking whether a behavior of an improved model of TFSM is deterministic can not be solved as easy and straightforwardly as in the case of the initial model of TFSM.

It should be noticed that the property of deterministic behavior is very important in theory real-time machines. As it was said above, universality, inclusion and equivalence checking problems are undecidable for Timed Automata in general case [2] but all these problems have been shown to be decidable for deterministic Timed Automata [3], [11]. However, testing whether a Timed Automaton is determinable has been proved undecidable [8]. Understanding and coping with these weaknesses have attracted lots of research, and classes of timed automata have been exhibited, that can be effectively determinized [3], [12]. A generic construction that is

applicable to every Timed Automaton, and which, under certain conditions, yields a deterministic Timed Automaton, which is language-equivalent to the original timed automaton, has been developed in [4].

We studied the determinacy checking problem for improved TFSMs and present the results of our research in this paper. First, we offer a criterion to determine whether a given sequence of transition (trace) in a TFSM is steady, i.e. for any input timed word which activates this trace the letters of output words always follow in the same order (but, maybe, with different timestamps). Then, using this criterion we developed an exhaustive algorithm for checking the property of strict determinacy of TFSMs. This property means that every input timed word activates no more than one trace and all traces in a TFSM are steady. Exhaustive search, although been time consuming, can hardly be avoided in this case: we proved that determinacy checking problem for improved version of TFSMs is co-NP-hard by polynomially reducing to its complement the subset-sum problem [7] which is known to be NP-complete.

The structure of the paper is as follows. In Section II we define the basic notions and introduce an improved concept of TFSM (or, it would be better said, a concept of TFSM with an improved behavior). In Section III we present necessary and sufficient conditions for steadiness of traces in a TFSM and show how to use this criterion to check whether a given TFSM is strictly deterministic. Section IV contains the results on the complexity of checking the properties of strictly deterministic behavior of TFSM. In the Conclusion we briefly outline the consequences of our results and topics for further research.

2. Formatting overview

Consider two non-empty finite alphabets I and O; the alphabet I is an *input alphabet* and the alphabet O is an *output alphabet*. The letters from I can be regarded as control signals received by some real-time computing system, whereas the letters from O may be viewed as responses (actions) generated by the system. A finite sequence $w = i_1, i_2, \dots, i_n$ of input letters is called an *input word*, whereas a sequence $z = o_1, o_2, \dots, o_n$ of output letters is called an *output word*. As usual, the time domain is represented by the set of non-negative reals $\mathbb{R}_0^+$. The set of all positive real numbers will be denoted by $\mathbb{R}^+$. When such a system receives a control signal (a letter i) its output depends not only on the input signal i but also on

- a current internal state of the system,
- a time instance when $\boldsymbol{i}$ becomes available to a system, and
- time required to process the input (output delay).

These aspects of real-time behavior can be formalized with the help of timestamps, time guards and delays. A timestamp as well as a delay is a real number from $\mathbb{R}^+$. A *timestamp* indicates a time instance when the system receives an input signal or generates a response to it. A *delay* is time the system needs to generate an output response after receiving an input signal. A *time guard* is an interval $g = \langle u, v \rangle$,

where $\langle \in \{(, [\}, \ \rangle \in \{),]\}$, and u, v are timestamps such that $0 < u < v$. Time intervals indicate the periods of time when transitions of a system are active for processing input signals. As usual, the term *time sequences* is reserved for an increasing sequence of timestamps. For the sake of simplicity we will deal only with time guards of the form $(u, v]$: all the results obtained in this paper can be adapted with minor changes to arbitrary time guards.

Let $\boldsymbol{w = x_1, x_2, \dots x_n}$ and $\boldsymbol{\tau = t_1, t_2, \dots, t_n}$ be an input (output) word and a time sequence, respectively, of the same length. Then a pair $(\boldsymbol{w, \tau})$ is called a *timed word*. Every pair of corresponding elements $\boldsymbol{x_j}$ and $\boldsymbol{t_j}$, $\boldsymbol{1 \le j \le n}$, indicates that an input signal (or an output response) $\boldsymbol{x_j}$ appears at time instance $\boldsymbol{t_j}$. In order to make this correspondence clearer we will often write timed words as sequences of pairs $\boldsymbol{(w, \tau) = (i_1, t_1), (i_2, t_2), \dots, (i_n, t_n)}$ whose components are input signals (or output responses) and their timestamps.

A *Finite State Machine* (*FSM*) over the alphabets $\boldsymbol{I}$ and $\boldsymbol{O}$ is a triple $\boldsymbol{M = \langle S, s_{in}, \rho \rangle}$ where $\boldsymbol{S}$ is a finite non-empty set of *states*, $\boldsymbol{s_{in}}$ is an *initial state*, $\boldsymbol{\rho \subseteq (S \times I \times O \times S)}$ is a *transition* relation. A transition $\boldsymbol{(s, i, o, s')}$ means that FSM $\boldsymbol{M}$ when being at the state $\boldsymbol{s}$ and receiving an input signal $\boldsymbol{i}$ moves to the state $\boldsymbol{s'}$ and generates the output response $\boldsymbol{o}$.

FSMs can not measure time and, therefore, they are unsuitable for modeling the behavior of real-time systems. The authors of [1] proposed to equip FSMs with clocks — variables which take non-negative real values. To manipulate with clocks machines use reset instructions, timed guards and output delays. Time guards indicate time intervals when transitions are active for processing input signals. An output delay indicates how much time does it take to process an input. Thus, every transition in such a machine is a quadruple $\boldsymbol{\langle input, timed\ guard, output, delay \rangle}$. Input signals and output responses are accompanied by timestamps. If an *input* is marked by a timestamp which satisfies the *time guard* then the transition fires, the machine moves to the next state and generates the *output*. This output is marked by a timestamp which is equal to the timestamp of the input plus the *delay*. For real-time machines of this kind usual problems from automata theory (equivalence and containment checking, minimization, etc.) may be set up and solved. The minimization problem for real-time machines is very important, since the complexity of many analysis and synthesis algorithms depend on the size of machines. In [14] this problem was studied under the so called "slow environment assumption": next input becomes available only after an output response to the previous one is generated.

In this paper, we consider a more advanced real-time machine; in this model the order in which outputs become visible to an outside observer is determined not only by the order in which inputs follow, but also by the delay required for their processing. When the same sequence of transitions is performed by such a machine the same outputs may follow in different order depending on the arriving time of the corresponding inputs. Our main goal is to develop equivalence checking and

minimization algorithms for real-time machines of this kind. But, as the results of Automata Theory show, these problems may have efficient solution only for deterministic machines. Thus, our first step toward the solution of these problems is to find a way to check if the behavior of a machine is deterministic.

But there is also another reason to study the problem of checking the determinism of the behavior of real-time machines. Unlike traditional discrete models of computation, the behavior of real-time machines depends not only on the control signals as such, but also on the time of their arrival. However, the latter factor has a greater degree of uncertainty. In most cases, in practice, it is desirable to reduce the effect of this uncertainty to a minimum. Therefore, the determinacy checking problem for real-time machines can be considered as a special version of the verification problem — checking that the time factor does not have an unforeseen influence on the behavior of the system.

Formally, by Timed FSM (TFSM) over the alphabets $\boldsymbol{I}$ and $\boldsymbol{O}$ we mean a quadruple $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{s_{in}}, \boldsymbol{G}, \boldsymbol{\rho})$ where:

- $\boldsymbol{S}$ is a finite non-empty *set of states*,
- $\boldsymbol{s_{in}}$ is *an initial state*.
- $\boldsymbol{G}$ is a set of *timed guards*,
- $\boldsymbol{\rho} \subseteq (\boldsymbol{S} \times \boldsymbol{I} \times \boldsymbol{O} \times \boldsymbol{S} \times \boldsymbol{G} \times \mathbb{R}^+)$ is a *transition relation*.

A transition (s, i, o, s', g, d) should be understood as follows. Suppose that TFSM receives the input letter i marked by a timestamp t when being at the state s. If the previous letter has been delivered to the TFSM at time $\hat{t}$ such that $\Delta t = t - \hat{t} \in g$ then the TFSM moves to the state s' and outputs the letter o marked with the timestamp $\tau = t + d$. When algorithmic and complexity issues of TFSM's analysis and synthesis are concerned then we assume that time guards and delays are rational numbers, and the size of a TFSM is the length of a binary string which encodes all transitions in the TFSM.

A *trace* $\boldsymbol{tr}$ in TFSM $\boldsymbol{M}$ is a sequence of transitions $(\boldsymbol{s_0}, \boldsymbol{a_1}, \boldsymbol{b_1}, \boldsymbol{s_1}, (\boldsymbol{u_1}, \boldsymbol{v_1}], \boldsymbol{d_1}), \dots, (\boldsymbol{s_{n-1}}, \boldsymbol{a_n}, \boldsymbol{b_n}, \boldsymbol{s_n}, (\boldsymbol{u_n}, \boldsymbol{v_n}], \boldsymbol{d_n})$, where every state $\boldsymbol{s_j}$, $\boldsymbol{0 < j < n}$, is an arrival state of one transition and a departure state of the next transition. We say that the trace $\boldsymbol{tr}$ *converts* an input timed word $\boldsymbol{\alpha} = (\boldsymbol{a_1}, \boldsymbol{t_1}), (\boldsymbol{a_2}, \boldsymbol{t_2}), \dots, (\boldsymbol{a_n}, \boldsymbol{t_n})$ to the timed output word $\boldsymbol{\beta} = (\boldsymbol{b_{j_1}}, \boldsymbol{\tau_1}), (\boldsymbol{b_{j_2}}, \boldsymbol{\tau_2}), \dots, (\boldsymbol{b_{j_n}}, \boldsymbol{\tau_n})$, iff

- $t_j - t_{j-1} \in (u_j, v_j]$ holds for all $j, 1 \le i \le n$ (it is assumed that $t_0 = 0$);
- β is such a permutation of the sequence $\gamma = (b_1, t_1 + d_1), (b_2, t_2 + d_2), \dots, (b_n, t_n + d_n)$ that the second components of the pairs $\tau_1, \tau_2, \dots, \tau_n$ constitute a time sequence.

Clearly, for every trace tr and an input timed word α its conversion β (if any) is determined uniquely; such a conversion will be denoted as $conv(tr, \alpha)$. If

$conv(tr, \alpha)$ is defined then we say that the input timed word α *activates* the trace tr. We will say that the output word $b_{j_1}, b_{j_2}, \ldots, b_{j_n}$ is a *plain response* to the input timed word α on the trace tr; it will be denoted as $resp(tr, \alpha)$.

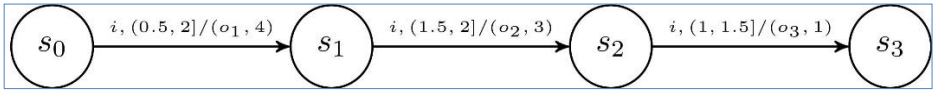


Fig.1 TFSM $\boldsymbol{M}$

Consider, for example, a TFSM $\boldsymbol{M}$ depicted in Fig. 1 and a trace

$$\boldsymbol{tr} = (\boldsymbol{s_0}, \boldsymbol{i}, \boldsymbol{s_1}, \boldsymbol{o_1}, (\boldsymbol{0.5}, \boldsymbol{2}], \boldsymbol{4}), (\boldsymbol{s_1}, \boldsymbol{i}, \boldsymbol{s_2}, \boldsymbol{o_2}, (\boldsymbol{1.5}, \boldsymbol{2}], \boldsymbol{3}), (\boldsymbol{s_2}, \boldsymbol{i}, \boldsymbol{s_3}, \boldsymbol{o_3}, (\boldsymbol{1}, \boldsymbol{1.5}], \boldsymbol{1})$$

in this TFSM. Then this trace

1. accepts an input timed word $\alpha_1 = (i, 1), (i, 2.7), (i, 4.1)$ and converts it to the output timed word $\beta_1 = (o_1, 5), (o_3, 5.1), (o_2, 5.7)$; thus, the plain response of M to α_1 is $w_1 = o_1, o_3, o_2$;
2. accepts an input timed word $\alpha_2 = (i, 1.5), (i, 3.2), (i, 4.3)$ and converts it to the output timed word $\beta_2 = (o_3, 5.3), (o_1, 5.5), (o_2, 6.2)$, and the plain response of M to α_2 is $w_2 = o_3, o_1, o_2$ which is different from w_1;

does not accept an input timed word $\alpha_3 = (i, 2.3), (i, 4), (i, 6)$.

3. Steady traces and strictly deterministic TFSMs

As can be seen from the above example, a pair of input timed words that differ only in timestamps of input signals may activate the same trace in a TFSM, although plain responses of TFSM to these words are different. Generally speaking, there is nothing unusual in this: in real-time models not only the input signals, but also the values of timers influence a run of a model. Nevertheless, in many applications it is critically important to be sure that the behavior of a real-time system is predictable: once a system choose a mode of computation (i.e. a trace in TFSM) it will behave in a similar way (i.e. give the same plain response) in all computations of this mode. Traditionally, computer systems in which for any input data the processing mode is uniquely determined by the system are called deterministic. But for our model of real-time systems this requirement should be clarified and strengthened. For this purpose, we introduce the notion of steady traces and the property of strict determinacy of a real-time system.

A trace $\boldsymbol{tr}$ in TFSM $\boldsymbol{M}$ is called *steady* if $\boldsymbol{resp(tr, \alpha_1) = resp(tr, \alpha_2)}$ holds for every pair of input timed words $\boldsymbol{\alpha_1}$ and $\boldsymbol{\alpha_2}$ that activate $\boldsymbol{tr}$. Thus, the order of the output letters generated by a steady trace does not depend on the small deviations of the timestamps of the input signals. A TFSM $\boldsymbol{M = (S, s_{in}, G, \rho)}$ is called *deterministic* iff for every pair of transitions $(\boldsymbol{s, i_1, o_1, s'}, (\boldsymbol{u_1, v_1}], \boldsymbol{d_1})$ and $(\boldsymbol{s, i_2, o_2, s''}, (\boldsymbol{u_2, v_2}], \boldsymbol{d_2})$ in $\boldsymbol{\rho}$ either $\boldsymbol{i_1 \neq i_2}$, or $(\boldsymbol{u_1, v_1}] \cap (\boldsymbol{u_2, v_2}] = \emptyset$. This requirement means that every timestamped input letter can activate no more than

one transition from an arbitrary given state $\boldsymbol{s}$. It also implies that every input timed word can activate no more than one trace in $\boldsymbol{M}$. A deterministic TFSM is called *strictly deterministic* iff every initial trace in $\boldsymbol{M}$ which starts from the initial state $\boldsymbol{s_{in}}$ is steady. It is easy to see that TFSM, depicted in Fig. 1, is not strictly deterministic.

The Strict Determinacy Checking Problem (in what follows, SDCP) is that of checking, given a TFSM, if it is strictly deterministic. It is easy to check whether a TFSM is deterministic by considering one by one all pairs of transitions that emerge from the same state. But local means alone are not enough to check whether a given trace in a TFSM is steady. A simple criterion for steadiness of traces is presented as a Theorem below.

Let a sequence of transitions

$$(\boldsymbol{s_0}, \boldsymbol{i_1}, \boldsymbol{s_1}, \boldsymbol{o_1}, \langle \boldsymbol{u_1}, \boldsymbol{v_1} \rangle, \boldsymbol{d_1}), \dots, (\boldsymbol{s_{n-1}}, \boldsymbol{i_n}, \boldsymbol{s_n}, \boldsymbol{o_n}, \langle \boldsymbol{u_n}, \boldsymbol{v_n} \rangle, \boldsymbol{d_n})$$

be a trace $\boldsymbol{tr}$ in a TFSM $\boldsymbol{M}$. Then the following theorem holds.

Theorem 1. *A trace $\boldsymbol{tr}$ is steady iff for all pairs of integers $\boldsymbol{k}, \boldsymbol{m}$ such that $\boldsymbol{1 \le k < m \le n}$ at least one of the two inequalities $\boldsymbol{d_k - d_m \le \sum_{j=k+1}^{m} u_j}$ or $\boldsymbol{d_k - d_m > \sum_{j=k+1}^{m} v_j}$ holds.*

Proof. (⇒) Suppose that there exists a pair $\boldsymbol{k, m}$ such that $\boldsymbol{1 \le k < m \le n}$, and a double inequality holds:

$$\sum_{j=k+1}^{m} \boldsymbol{u_j} < \boldsymbol{d_k - d_m} \le \sum_{j=k+1}^{m} \boldsymbol{v_j}\,.$$

Then we use two positive numbers $\boldsymbol{r = d_k - d_m - \sum_{j=k+1}^{m} u_j}$ and $\boldsymbol{\varepsilon = \frac{r}{n}}$ and consider a behaviour of a TFSM $\boldsymbol{M}$ in the input timed words

$$\boldsymbol{\alpha' = (i_1, v_1), \dots, (i_k, \sum_{j=1}^{k} v_j), (i_{k+1}, \sum_{j=1}^{k} v_j + u_{k+1} + \varepsilon), \dots, (i_m, \sum_{j=1}^{k} v_j + \sum_{j=k+1}^{m} u_j + \varepsilon),}$$

$$\boldsymbol{\alpha'' = (i_1, v_1), \dots, (i_k, \sum_{j=1}^{k} v_j), (i_{k+1}, \sum_{j=1}^{k+1} v_j), \dots, (i_m, \sum_{j=1}^{m} v_j).}$$

It is easy to see that both words activate $\boldsymbol{tr}$.

The trace $\boldsymbol{tr}$ converts the timed input word $\boldsymbol{\alpha_1}$ to the timed output word

$$\boldsymbol{conv(tr, \alpha') = \dots, (o_m, T'_m), \dots, (o_k, T'_k), \dots}$$

such that $\boldsymbol{T'_m = \sum_{j=1}^{k} v_j + \sum_{j=k+1}^{m} (u_j + \varepsilon) + d_m}$, and $\boldsymbol{T'_k = \sum_{j=1}^{k} v_j + d_k}$. In this timed output word, the output letter $\boldsymbol{o_k}$ follows the output letter $\boldsymbol{o_m}$ since

$$\boldsymbol{T'_k - T'_m = d_k - d_m - \sum_{j=k+1}^{m} u_j + (m-k)\varepsilon = r - \frac{r(m-k)}{n} > 0.}$$

Hence, $\boldsymbol{resp}(\boldsymbol{tr}, \boldsymbol{\alpha}') = \cdots, \boldsymbol{o_m}, \ldots, \boldsymbol{o_k}, \ldots$.

On the other hand, the trace $\boldsymbol{tr}$ converts the timed input word $\boldsymbol{\alpha}''$ to the timed output word

$$\boldsymbol{conv}(\boldsymbol{tr}, \boldsymbol{\alpha}'') = \cdots, (\boldsymbol{o_k}, \boldsymbol{T''_k}), \ldots, (\boldsymbol{o_m}, \boldsymbol{T''_m}), \ldots$$

such that $\boldsymbol{T''_k} = \sum_{j=1}^{k} \boldsymbol{v_j} + \boldsymbol{d_k}$ and $\boldsymbol{T''_m} = \sum_{j=1}^{m} \boldsymbol{v_j} + \boldsymbol{d_m}$. In this timed output word the output letter $\boldsymbol{o_m}$ follows the output letter $\boldsymbol{o_k}$ since

$$\boldsymbol{T''_m} - \boldsymbol{T''_k} = \boldsymbol{d_m} - \boldsymbol{d_k} = \sum_{j=k+1}^{m} \boldsymbol{v_j} \geq \boldsymbol{0}$$

Therefore, $\boldsymbol{resp}(\boldsymbol{tr}, \boldsymbol{\alpha}'') = \cdots, \boldsymbol{o_k}, \ldots, \boldsymbol{o_m}, \ldots$.

Thus, we got evidence that the trace $\boldsymbol{tr}$ is not steady.

($\Leftarrow$) Suppose that the trace $\boldsymbol{tr}$ is not steady. Then there exists a pair of timed input words $\boldsymbol{\alpha}' = (\boldsymbol{i_1}, \boldsymbol{t'_1}), \ldots, (\boldsymbol{i_n}, \boldsymbol{t'_n})$ and $\boldsymbol{\alpha}'' = (\boldsymbol{i_1}, \boldsymbol{t''_1}), \ldots, (\boldsymbol{i_n}, \boldsymbol{t''_n})$ such that both words activate the trace $\boldsymbol{tr}$ and $\boldsymbol{resp}(\boldsymbol{tr}, \boldsymbol{\alpha}') \neq \boldsymbol{resp}(\boldsymbol{tr}, \boldsymbol{\alpha}'')$. Consequently, there exists a pair of output letters $\boldsymbol{o_m}$ and $\boldsymbol{o_k}$ such that

$$\boldsymbol{conv}(\boldsymbol{tr}, \boldsymbol{\alpha}') = \cdots, (\boldsymbol{o_k}, \boldsymbol{T'_k}), \ldots, (\boldsymbol{o_m}, \boldsymbol{T'_m}), \ldots$$
$$\boldsymbol{conv}(\boldsymbol{tr}, \boldsymbol{\alpha}'') = \cdots, (\boldsymbol{o_m}, \boldsymbol{T''_m}), \ldots, (\boldsymbol{o_k}, \boldsymbol{T''_k}), \ldots .$$

Such permutation of output letters is possible iff the following inequalities hold

$$\boldsymbol{t'_k} + \boldsymbol{d_k} = \boldsymbol{T'_k} < \boldsymbol{T'_m} = \boldsymbol{t'_m} + \boldsymbol{d_m},$$
$$\boldsymbol{t''_k} + \boldsymbol{d_k} = \boldsymbol{T''_k} > \boldsymbol{T''_m} = \boldsymbol{t''_m} + \boldsymbol{d_m}.$$

But since both input timed words $\boldsymbol{\alpha}'$ and $\boldsymbol{\alpha}''$ activate $\boldsymbol{tr}$, we have the following chain of inequalities:

$$\sum_{j=k+1}^{m} \boldsymbol{u_j} < \boldsymbol{T''_m} - \boldsymbol{T''_k} < \boldsymbol{d_k} - \boldsymbol{d_m} < \boldsymbol{T'_m} - \boldsymbol{T'_k} \leq \sum_{j=k+1}^{m} \boldsymbol{v_j}.$$

Thus, if $\boldsymbol{tr}$ is not steady then there exists a pair of integers such that $\boldsymbol{1} \leq \boldsymbol{k} < \boldsymbol{m} \leq \boldsymbol{n}$ and

$$\sum_{j=k+1}^{m} \boldsymbol{u_j} < \boldsymbol{d_k} - \boldsymbol{d_m} \leq \sum_{j=k+1}^{m} \boldsymbol{v_j}$$

holds.

End proof.

Now, having the criterion for steadiness of traces, we can give a solution to SDCP for TFSMs. Let TFSM $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{s_{in}}, \boldsymbol{G}, \boldsymbol{\rho})$ be a deterministic TFSM. Denote by $\boldsymbol{u_{min}}$ the greatest lower bound of all left boundaries used in the time guards of $\boldsymbol{M}$. In our model of TFSM $\boldsymbol{u_{min}} > \boldsymbol{0}$. Let $\boldsymbol{d_{min}}$ and $\boldsymbol{d_{max}}$ be the minimum and the maximum

output delays occurred in the transitions of $\boldsymbol{M}$. A theorem below gives necessary and sufficient conditions for the behaviour of $\boldsymbol{M}$ to be strictly deterministic.

Theorem 2. *A deterministic TFSM* $\boldsymbol{M}$ *is strictly deterministic iff all its traces of length* $\boldsymbol{p}$*, where* $\boldsymbol{p} = \lceil \frac{d_{max} - d_{min}}{u_{min}} \rceil$*, are steady.*

Proof. The necessity of conditions is obvious.

We prove the sufficiency of conditions by contradiction. Suppose that all traces of length less or equal $\boldsymbol{p}$ are steady but TFSM $\boldsymbol{M}$ is not. Then there exists such a trace $\boldsymbol{tr}$ in $\boldsymbol{M}$ which is not steady. Then, by Theorem 1, this trace is a sequence of transitions $(s_{j-1}, i_j, s_j, b_j, (u_j, v_j], d_j), 1 \le j \le n$, such that for some pair of integers $\boldsymbol{m}$ and $\boldsymbol{k}$, where $1 \le k < m \le n$, two inequalities

$$\sum_{j=k+1}^{m} u_j \le d_k - d_m \le \sum_{j=k+1}^{m} v_j$$

hold. It should be noticed, that, by the same Theorem 1, the trace $\boldsymbol{tr'}$ which includes only the transitions $(s_{j-1}, i_j, s_j, b_j, (u_j, v_j], d_j), m \le j \le k$, is not steady as well. Hence, $m - k > p$, and we have the following sequence of inequalities

$$d_{max} - d_{min} \ge d_m - d_k \ge \sum_{j=k+1}^{m} u_j > p * u_{min}$$

which contradicts our choice of $\boldsymbol{p} = \lceil \frac{d_{max} - d_{min}}{u_{min}} \rceil$.

End of proof.

As it follows from Theorems 1 and 2, to guarantee that a given TFSM $\boldsymbol{M} = (S, s_{in}, G, \rho)$ is strictly deterministic it is sufficient to consider all traces $(s_0, a_1, b_1, s_1, (u_1, v_1], d_1), \dots, (s_{n-1}, a_n, b_n, s_n, (u_n, v_n], d_n)$ in $\boldsymbol{M}$, whose length $\boldsymbol{n}$ does not exceed the value $\boldsymbol{p} = \lceil \frac{d_{max} - d_{min}}{u_{min}} \rceil$ defined in Theorem 2, and for every such trace check that one of the inequalities $d_1 - d_n < \sum_{j=2}^{n} u_j$ or $d_1 - d_n > \sum_{j=2}^{n} v_j$ holds. Thus, we arrive at

Corollary 1. *Strict Determinacy Checking Problem for TFSMs is decidable.*

4. Strict Determinacy Checking Problem for TFSMs is co-NP-hard

Clearly, the decision procedure, based on Theorem 2, is time consuming since p may be exponential of the size of M and the number of traces of length p in TFSM M is exponential of p. In this section we show that such an exhaustive search can hardly be avoided because SDCP for improved version of TFSMs is co-NP-hard.

We are aimed to show that the complement of SDCP is NP-hard. To this end we consider the Subset-Sum Problem (see [7]) which is known to be NP-complete and

demonstrate that this problem can be reduced in polynomial time to the complement of SDCP for TFSMs.

The Subset-Sum Problem (SSP) is that of checking, given a set of integers $\boldsymbol{Q}$ and an integer $\boldsymbol{L}$, whether there is any subset $\boldsymbol{Q'}, \boldsymbol{Q'} \subseteq \boldsymbol{Q}$, such that the sum of all its elements is equal to $\boldsymbol{L}$. More formally, the variant of the SSP we are interested in is defined as follows. Let $\boldsymbol{Q} = \boldsymbol{m_1}, \boldsymbol{m_2}, \dots, \boldsymbol{m_N}$ be a sequence of positive integers, and $\boldsymbol{L}$ be also a positive integer. A solution to $(\boldsymbol{Q}, \boldsymbol{L})$-instance of SSP is a binary tuple $\boldsymbol{z} = \langle \boldsymbol{\sigma_1}, \boldsymbol{\sigma_2}, \dots, \boldsymbol{\sigma_N} \rangle$ such that $\sum_{j=1}^{N} \boldsymbol{\sigma_j}\, \boldsymbol{m_j} = \boldsymbol{L}$. In [7] it was proved that the problem of checking the existence of a solution to a given $(\boldsymbol{Q}, \boldsymbol{L})$-instance of SSP is NP-complete.

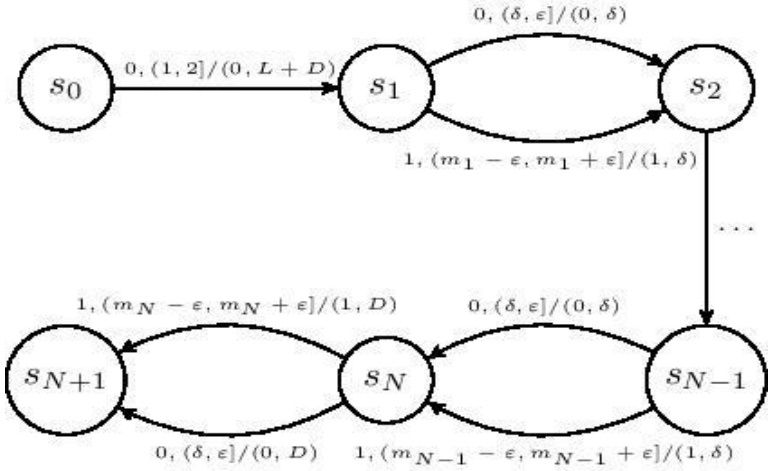


Fig.2 TFSM M

Now, given a $(\boldsymbol{Q}, \boldsymbol{L})$-instance of SSP, we show how to build a deterministic TFSM $\boldsymbol{M_{Q,L}}$ such that it has an initial trace which is *not* strictly determined iff this instance of SSP has a solution. Let $\boldsymbol{D} = \sum_{j=1}^{N} \boldsymbol{m_j}$, and $\boldsymbol{\varepsilon}$ and $\boldsymbol{\delta}$ be positive rational numbers such that $\boldsymbol{\varepsilon} = \boldsymbol{o}(\boldsymbol{1}/\boldsymbol{N^2})$ and $\boldsymbol{\delta} = \boldsymbol{o}(\boldsymbol{\varepsilon}/\boldsymbol{N^2})$. Consider a TFSM depicted in Fig. 2. This machine operates over alphabets $\boldsymbol{I} = \boldsymbol{O} = \{\boldsymbol{0}, \boldsymbol{1}\}$. It has $\boldsymbol{N + 2}$ states $\boldsymbol{s_0}, \boldsymbol{s_1}, \dots, \boldsymbol{s_n}, \boldsymbol{s_{N+1}}$. The only transition $(\boldsymbol{s_0}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{s_1}, (\boldsymbol{1}, \boldsymbol{2}], \boldsymbol{L + D})$ leads from the initial state $\boldsymbol{s_0}$ to $\boldsymbol{s_1}$. From each state $\boldsymbol{s_j}, \boldsymbol{1 \le j < N}$, two transitions $(\boldsymbol{s_j}, \boldsymbol{1}, \boldsymbol{1}, \boldsymbol{s_{j+1}}, (\boldsymbol{m_j - \varepsilon}, \boldsymbol{m_j + \varepsilon}], \boldsymbol{\delta})$ and $(\boldsymbol{s_j}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{s_{j+1}}, (\boldsymbol{\delta}, \boldsymbol{\varepsilon}], \boldsymbol{\delta})$ lead to the state $\boldsymbol{s_{j+1}}$. The state $\boldsymbol{s_N}$ is different: two transitions $(\boldsymbol{s_N}, \boldsymbol{1}, \boldsymbol{1}, \boldsymbol{s_{N+1}}, (\boldsymbol{m_N - \varepsilon}, \boldsymbol{m_N + \varepsilon}], \boldsymbol{D})$ and $(\boldsymbol{s_N}, \boldsymbol{0}, \boldsymbol{0}, \boldsymbol{s_{N+1}}, (\boldsymbol{\delta}, \boldsymbol{\varepsilon}], \boldsymbol{D})$ lead this state to $\boldsymbol{s_{N+1}}$.

First, we make some observations.

1) Since all transitions outgoing from the states $\boldsymbol{s_j}, \boldsymbol{1 \le j < N}$, have the same delay $\boldsymbol{\delta}$, every trace from a state $\boldsymbol{s_k}$ to a state $\boldsymbol{s_\ell}$, where $\boldsymbol{0 < k < \ell \le N}$, is strictly deterministic.

2) Since $\boldsymbol{\delta = o(1/N^4)}$ and $\boldsymbol{0 < \varepsilon = o(1/N^2)}$, for every $\boldsymbol{k, 1 < k \le N}$, and a binary tuple $\boldsymbol{z = \langle \sigma_k, \sigma_{k+1}, \dots, \sigma_N \rangle}$ the inequalities

$$\boldsymbol{\delta - D < 0 < N\delta \le \sum_{j=k+1}^{N} (\sigma_j(m_j - \varepsilon) + (1 - \sigma_j)\delta)}$$

hold. By Theorem 1, this implies that every trace from a state $\boldsymbol{s_k, 1 \le k \le N}$, to the state $\boldsymbol{s_{N+1}}$ is strictly deterministic.

3) For the same reason the inequalities

$$\boldsymbol{D + L - \delta > \sum_{j=1}^{k} m_j \; + \; k\varepsilon = \sum_{j=1}^{k} (\sigma_j(m_j + \varepsilon) + (1 - \sigma_j)\varepsilon)}$$

hold for every $\boldsymbol{k, 1 \le k < N}$, and a binary tuple $\boldsymbol{z = \langle \sigma_1, \sigma_2, \dots, \sigma_k \rangle}$. By Theorem 1, this guarantees that every initial trace leading to a state $\boldsymbol{s_k, 1 \le k \le N}$ is strictly deterministic.

As for the initial traces that lead to the state $\boldsymbol{s_{N+1}}$, due to our choice of $\boldsymbol{\varepsilon}$ and $\boldsymbol{\delta}$, we can trust the following chain of reasoning. By definition, a $\boldsymbol{(Q, L)}$-instance of SSP has a solution $\boldsymbol{z = \langle \sigma_1, \sigma_2, \dots, \sigma_N \rangle}$ iff $\boldsymbol{\sum_{j=1}^{N} \sigma_j m_j = L}$. The latter is possible iff two following inequalities hold:

$$\boldsymbol{\sum_{j=1}^{N} \sigma_j m_j \; - \; \varepsilon + N\delta < L < \sum_{j=1}^{N} \sigma_j (m_j) \; + N\varepsilon} \qquad (1)$$

By taking into account the relationships below

$$\boldsymbol{\sum_{j=1}^{N} (\sigma_j(m_j - \varepsilon) + (1 - \sigma_j)\delta) < \sum_{j=1}^{N} \sigma_j m_j \; - \; \varepsilon + N\delta}$$

$$\boldsymbol{\sum_{j=1}^{N} \sigma_j (m_j) \; + N\varepsilon = \sum_{j=1}^{N} (\sigma_j(m_j + \varepsilon) + (1 - \sigma_j)\varepsilon),}$$

we can conclude that (1) holds iff another pair of inequalities hold:

$$\boldsymbol{\sum_{j=1}^{N} (\sigma_j(m_j - \varepsilon) + (1 - \sigma_j)\delta) < L < \sum_{j=1}^{N} (\sigma_j(m_j + \varepsilon) + (1 - \sigma_j)\varepsilon)}$$

But in the context of observations 1) – 3) above, the latter inequalities, as it follows from Theorem 1, provide the necessary and sufficient conditions that the initial trace in TFSM $\boldsymbol{M_{Q,L}}$ activated by the input word $\boldsymbol{z = \langle \sigma_1, \sigma_2, \dots, \sigma_N \rangle}$ is not strictly deterministic.

Thus, a $\boldsymbol{(Q, L)}$-instance of SSP has a solution iff TFSM $\boldsymbol{M_{Q,L}}$ is not strictly deterministic.

The considerations above bring us to

Theorem 3. *SDCP for TFSMs is co-NP-hard.*

5. Conclusion

The main contributions of this paper are

1. the development of a modified version of TFSM which, in our opinion, provides a more adequate model of real-time computing systems;
2. the introduction of the notion of strict deterministic behaviour of TFSM and setting up the Strict Determinacy Checking Problem (SDCP) for a modified version of TFSMs;
3. the establishing of an effectively verifiable criterion for the strict determinacy property of TFSMs;
4. the proving that SDCP for TFSMs is co-NP-hard.

However, some problems concerning strict deterministic behavior of TFSMs still remain open. They will be topics for our further research.

1. In Sections [Sect3] and [Sect4] it was shown that SDCP for TFSMs is co-NP-hard and in the worst case it can be solved in double exponential time by means of a naive exhaustive searching algorithm based on Theorems 1 and 2. We think that this complexity upper bound estimate is too much high. The question arises, for what complexity class $\mathscr{C}$ SDCP for TFSMs is a $\mathscr{C}$-complete problem. By some indications we assume that SDCP for TFSMs is PSPACE-complete problem.
2. As it can be seen from the proof of Theorem 3, SDCP for TFSMs is intractable only if timed parameters of transitions (time guards and delays) depend on the number of states in TFSM. But this is not a typical phenomenon in real-time systems since in practice the performance of individual components of a system does not depend on the size of the system. Therefore, it is reasonable to confine ourselves to considering only such TFSMs, in which the time guards and the delays are chosen from some fixed finite set. As it follows from Theorem 2, for this class of TFSMs SDCP is decidable in polynomial time. One may wonder what is the degree of such a polynomial, or, in other words, how efficiently the strict determinacy property can be checked for TFSMs corresponded to real systems.
3. In the model of TFSM besides the usual transitions there are also possible timeout transitions. A timeout transition fires when a timestamped input letter $(\boldsymbol{i}, \boldsymbol{t})$ can not activate any usual transition from a current state. In it was shown that in some cases such timeout transitions can not be replaced by any combination of ordinary transitions. In the future we are going to study how SDCP can be solved for TFSMs with timeouts.

Acknowledgments

The authors of the article express their deep gratitude to V.V. Podymov and the anonymous reviewers for their valuable comments and advice on improving the article. This work was supported by the Russian Foundation for Basic Research, Grant N 18-01-00854.

References

[1]. Alur R., Dill D. A Theory of Timed Automata. Theoretical Computer Science, vol. 126, 1994, pp. 183-235.

[2]. Alur R., Madhusudan P. Decision Problems for Timed Automata: A Survey. In Proceedings of the 4-th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'04), 2004, pp. 1-24.

[3]. Alur R., Fix L., Henzinger Th. A. A Determinizable Class of Timed Automata. In Proceedings of the 6-th International Conference on Computer Aided Verification (CAV'94), 1994, p 1-13.

[4]. Baier C., Bertrand N., Bouyer P., Brihaye T. When are Timed Automata Determinizable? In Proceedings of the 36-th International Colloquium on Automata, Languages, and Programming (ICALP 2009), 2009, p. 43-54.

[5]. Bresolin D., El-Fakih K., Villa T., Yevtushenko N. Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power. In Proceedings of the International Conference GANDALF, 2014, p. 203-216.

[6]. Cardell-Oliver R. Conformance Tests for Real-Time Systems with Timed Automata Specifications. Formal Aspects of Computing, vol. 12, no. 5, 2000, p. 350–371.

[7]. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. 35.5: The subset-sum problem. Introduction to Algorithms (2-nd ed.), 2001.

[8]. Finkel O. Undecidable Problems about Timed Automata. In Proceedings of 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06), 2006, p. 187-199.

[9]. Fletcher J. G., Watson R. W. Mechanism for Reliable Timer-Based Protocol. Computer Networks, vol. 2, 1978, pp. 271-290.

[10]. Merayo M.G., Nuunez M., Rodriguez I. Formal Testing from Timed Finite State Machines. Computer Networks, vol. 52, no 2, 2008, pp. 432-460.

[11]. Ouaknine J., Worrell J. On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In Proceedings of the 19-th Annual Symposium on Logic in Computer Science (LICS'04), 2004, pp. 54-63.

[12]. Suman P.V., Pandya P.K., Krishna S.N., Manasa L. Timed Automata with Integer Resets: Language Inclusion and Expressiveness. In Proceedings of the 6-th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08), 2008, pp. 78–92.

[13]. Tvardovskii A., Yevtushenko N. Minimizing Timed Finite State Machines. Tomsk State University Journal of Control and Computer Science, No 4 (29), 2014, pp. 77-83 (in Russian).

[14]. Tvardovskii A., Yevtushenko N., M. Gromov. Minimizing Finite State Machines with Time Guards and Timeouts. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 139-154 (in Russian).

[15]. Zhigulin M., Yevtushenko N., Maag S., Cavalli A. FSM-Based Test Derivation Strategies for Systems with Timeouts. In Proceedings of the 11-th International Conference on Quality Software, 2011, p. 141-149.

К проверке строго детерминированного поведения временных конечных автоматов

Е.М.Винарский <vinevg2015@gmail.com>
В.А. Захаров < zakh@cs.msu.su >.
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1

Аннотация. Конечные автоматы широко применяются в качестве математических моделей при решении многочисленных задач в области программирования, проектирования микроэлектронных схем и телекоммуникационных систем. Для описания поведения систем реального времени модель конечного автомата может быть расширена добавлением в неё часов - параметра непрерывного времени, моделируемого вещественной переменной. В автоматах реального времени для входных и выходных сигналов указывается время их поступления и выдачи, а переходы автомата снабжены описанием задержек, связанных с ожиданием входных сигналов и формированием выходных сигналов. Так же, как и для классических автоматов дискретного времени, задача минимизации конечных автоматов реального времени возникает во многих приложениях этой модели вычислений. Для классической модели автоматов реального времени эта задача уже подробно рассмотрена. В нашей работе мы предлагаем более сложную модель: в ней порядок следования выходных сигналов определяется не только порядком поступления входных сигналов, но также и задержкой, связанной с их обработкой. В этой модели при выполнении одной и той же последовательности переходов выходные сигналы могут выдаваться в разном порядке в зависимости от времени поступления входных сигналов. В новой модели автоматов реального времени решению задачи минимизации должно предшествовать изучение вопроса строгой детерминированности - однозначности поведения автомата на одних и тех же последовательностях переходов. В представленной статье приведены и обоснованы необходимые и достаточные условия строгой детерминированности автоматов реального времени, а также исследованы вопросы, связанные с решением задачи минимизации этой разновидности автоматов.

Ключевые слова: конечные временные автоматы; строго детерминированное поведение

DOI: 10.15514/ISPRAS-2018-30(3)-22

Для цитирования: Винарский Е.М., Захаров В.А. К проверке строго детерминированного поведения временных конечных автоматов. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 325-340 (на английском языке). DOI: 10.15514/ISPRAS-2018-30(3)-22

Список литературы

[1]. Alur R., Dill D. A Theory of Timed Automata. Theoretical Computer Science, vol. 126, 1994, pp. 183-235.

[2]. Alur R., Madhusudan P. Decision Problems for Timed Automata: A Survey. In Proceedings of the 4-th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'04), 2004, pp. 1-24.

[3]. Alur R., Fix L., Henzinger Th. A. A Determinizable Class of Timed Automata. In Proceedings of the 6-th International Conference on Computer Aided Verification (CAV'94), 1994, p 1-13.

[4]. Baier C., Bertrand N., Bouyer P., Brihaye T. When are Timed Automata Determinizable? In Proceedings of the 36-th International Colloquium on Automata, Languages, and Programming (ICALP 2009), 2009, p. 43-54.

[5]. Bresolin D., El-Fakih K., Villa T., Yevtushenko N. Deterministic Timed Finite State Machines: Equivalence Checking and Expressive Power. In Proceedings of the International Conference GANDALF, 2014, p. 203-216.

[6]. Cardell-Oliver R. Conformance Tests for Real-Time Systems with Timed Automata Specifications. Formal Aspects of Computing, vol. 12, no. 5, 2000, p. 350–371.

[7]. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. 35.5: The subset-sum problem. Introduction to Algorithms (2-nd ed.), 2001.

[8]. Finkel O. Undecidable Problems about Timed Automata. In Proceedings of 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06), 2006, p. 187-199.

[9]. Fletcher J. G., Watson R. W. Mechanism for Reliable Timer-Based Protocol. Computer Networks, vol. 2, 1978, pp. 271-290.

[10]. Merayo M.G., Nuunez M., Rodriguez I. Formal Testing from Timed Finite State Machines. Computer Networks, vol. 52, no 2, 2008, pp. 432-460.

[11]. Ouaknine J., Worrell J. On the Language Inclusion Problem for Timed Automata: Closing a Decidability Gap. In Proceedings of the 19-th Annual Symposium on Logic in Computer Science (LICS'04), 2004, pp. 54-63.

[12]. Suman P.V., Pandya P.K., Krishna S.N., Manasa L. Timed Automata with Integer Resets: Language Inclusion and Expressiveness. In Proceedings of the 6-th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08), 2008, pp. 78–92.

[13]. А.С. Твардовский, Н.В. Евтушенко. К минимизации автоматов с временными ограничениями. Вестник Томского государственного университета. Управление, вычислительная техника и информатика, vol. 29, no 4, 2014, pp. 77-83.

[14]. Твардовский А.С., Евтушенко Н.В., Громов М.Л. Минимизация автоматов с таймаутами и временными ограничениями. Труды ИСП РАН, том 29, вып. 4, 2017 г., стр. 139-154. DOI: 10.15514/ISPRAS-2017-29(4)-8..

[15]. Zhigulin M., Yevtushenko N., Maag S., Cavalli A. FSM-Based Test Derivation Strategies for Systems with Timeouts. In Proceedings of the 11-th International Conference on Quality Software, 2011, p. 141-149.

К построению модульной модели распределенного интеллекта

[1,2]*Ю.Л. Словохотов <slov@phys.chem.msu.ru>*
[3]*И.С. Неретин <ivan@neretin.ru>*
[1]*Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[2]*Институт элементоорганических соединений РАН, 119991, Россия, Москва, ул. Вавилова, д. 28*
[3]*Rock Flow Dynamics, 117418, Россия, Москва, ул. Профсоюзная, д. 25А*

Аннотация. Описание и моделирование динамики мультиагентных социальных систем методами, заимствованными из статистической физики «неживых» многочастичных систем, не отражает принципиальную особенность совокупности взаимодействующих автономных агентов: способность воспринимать, обрабатывать и использовать внешнюю информацию. Распределенный интеллект социальных систем следует непосредственно учитывать в их экспериментальных и теоретических исследованиях. В работе предложена «модульная» модель интеллектуальной деятельности, включающая производство новой информации и пригодная для описания как индивидуального, так и распределенного интеллекта, перечислены возможные области ее использования. «Количественную оценку» эффективности распределенного интеллекта иллюстрирует компьютерная модель искусственной социальной системы; обсуждаются полученные результаты.

Ключевые слова: мультиагентные социальные системы; распределенный интеллект; моделирование интеллектуальной деятельности

DOI: 10.15514/ISPRAS-2018-30(3)-23

Для цитирования: Словохотов Ю.Л., Неретин И.С. К построению модульной модели распределенного интеллекта. Труды ИСП РАН, том 30, вып. 3, 2018 г., стр. 341-362. DOI: 10.15514/ISPRAS-2018-30(3)-23

1. Введение

Изучению и прогнозированию динамики реально существующих мультиагентных социальных систем – таких как пешеходные потоки [1], объекты управления [2], участники политического противоборства [3], избирательные кампании [4] и многих других – посвящена обширная литература. Основным формальным инструментом анализа служит теория

игр, призванная типологизировать поведение агентов[1] и установить возможные равновесия в системе. Структуру совокупности взаимосвязанных агентов исследуют методами теории графов, представляя достаточно большие системы в виде сложных сетей [5]. Это позволяет выделить характерные структурные элементы (концентраторы, клики, сообщества, деревья), определить количественные характеристики сети (диаметр, распределение порядков вершин, степень кластеризации и др.), а также моделировать динамику различных процессов на сетевых субстратах [6–8].

При большом количестве положительных результатов и новых эмпирических наблюдений, в теоретико-игровых и теоретико-графовых исследованиях мультиагентных систем до сих пор не выработано их общепринятой модели. Моделирование мультиагентных процессов методами теории игр сталкивается с необходимостью учитывать как многоуровневую рефлексию агентов, обладающих интеллектом [9], так и «ограниченно рациональное» поведение людей в реальных условиях [10]. Это усложняет модели, делает их стохастическими и зависящими от эмпирических параметров, снижая предсказательную способность. Даже для систем с простыми стратегиями агентов, которые удается выразить квази-динамическими уравнениями, расчетный краткосрочный прогноз требует больших вычислительных мощностей (транспортный поток на шоссе [11]) либо оказывается невозможным за пределами нескольких идеализированных режимов (биржа [12]). С другой стороны, результаты моделирования социальных процессов на сетях во многих случаях показывают отсутствие сильного влияния структуры субстрата (которое неявно постулируется в большинстве «сетевых» работ). Так, основные качественные результаты математической социологии, полученные во 2-й половине XX века на квадратных решетках, изображавших систему социальных связей [13, 14], в целом воспроизводятся и на сложные сетях; различия в основном состоят в замедлении процесса и неоднородных «промежуточных» распределениях состояний агентов [14, 15].

С формальной точки зрения, большинство современных мультиагентных моделей построены по аналогии с моделями многочастичных физических систем [14]. Подобно взаимодействиям в ансамбле «неживых» частиц, взаимодействия агентов основаны на балансе количественно или качественно оцениваемых факторов (выигрышей, намерений, интересов и т.д.), а динамика каждого агента направляется стремлением к максимуму энергоподобной функции полезности и «размывается» стохастическим шумом, нередко называемым температурой. Усложнения агентных моделей, по сравнению с их физическими прототипами, сводятся к использованию нечетко определенных характеристик (начиная с «полезности»), неоднозначных зависимостей

[1] В рамках этой работы понятия «индивид», «индивидуум» и «агент» применительно к единичному «актору» социальной системы используются как синонимы.

(рефлексии) и сложной дискретной конфигурации взаимодействий (сети). Предсказуемый результат состоит в сильном увеличении объема расчетов и неопределенности прогноза в таких системах, где динамика процесса не задается самой сетевой структурой (как в эпидемиологии, энергетических и транспортных сетях, некоторых других областях).

Имеющиеся результаты мультиагентного моделирования позволяют предположить, что многочастичные модели, заимствованные из статистической термодинамики и физики стохастических процессов, не вполне адекватны фундаментальным особенностям социальных систем и процессов в таких системах. (Утверждение о несводимости социума к физике является общим местом в работах гуманитарного направления). Интригующим дополнительным обстоятельством служат эмпирические аналогии коллективного поведения людей и динамики, наблюдаемой для систем индивидов со значительно меньшими интеллектуальными возможностями (муравейник, термитник [16]), а также для формаций программируемых автоматов [17]. Таким образом, наличие индивидуального сознания у людей не является принципиальным отличием социальной мультиагентной системы от физической системы «неживых» частиц (молекул и атомов при сколь угодно детальном описании их состояний и взаимодействий): сходную динамику могут проявлять социальные системы, сильно различающиеся по уровню интеллекта у их агентов.

Мы предполагаем, что главной особенностью, отличающей систему взаимодействующих агентов от ансамбля частиц физической среды, является распределенный интеллект: способность мультиагентной системы воспринимать внешнюю информацию, обрабатывать и использовать ее в своей коллективной динамике. Наличие распределенного интеллекта следует непосредственно учитывать как в эмпирическом описании социальных мультиагентных систем, так и в их моделировании. Далее будут рассмотрены некоторые возможные подходы к исследованиям в этом направлении.

2. Определения

Мультиагентная социальная система (МСС) – это динамическая совокупность автономных агентов, которые воспринимают информацию и взаимодействуют с внешней средой и другими агентами в ходе собственной деятельности (*динамики*). Взаимодействующие биологические существа одного вида составляют социальную систему в узком смысле слова. «Неживые» программируемые агенты (роботы, беспилотные аппараты и т.д.) при наличии взаимодействия между ними образуют искусственную МСС. Индивидуальные агенты составляют МСС 1-го уровня (сообщество живых организмов, автомобили на шоссе и т.д.). Взаимодействующие мультиагентные системы (экономические субъекты, научные школы, политические «акторы») сами могут выступать в роли агентов в социальной системе более высокого уровня.

Каждая МСС (1) обладает структурой, (2) осуществляет некоторую суммарную («системную») деятельность, (3) в ее ходе воспринимает, обрабатывает и использует информацию. Структуру МСС определяют воздействия окружающей среды на агентов («внешнее поле») и взаимодействия между агентами (внутренняя структура, в общем случае – взвешенный ориентированный граф, зависящий от внешнего поля и от времени). Динамика МСС определяется изменениями внешней среды, собственной динамикой агентов и эволюцией внутренней структуры во времени.

Системная динамика МСС складывается из индивидуальных динамик агентов, однако ее результаты не сводятся к сумме индивидуальных результатов агентов (перемещение стада животных из засушливой местности в благоприятную, биржевой крах, автоматизация производства и т.д.). Поскольку агенты-индивиды в МСС 1-го уровня воспринимают информацию и преследуют определенный набор целей, социальным системам как «акторам» можно приписать *системное целеполагание*. Объективной целью МСС на заданном интервале времени является оптимальный для системы результат ее динамики. Разнообразные МСС с вариативной динамикой агентов, не предполагающей сложной интеллектуальной деятельности, воспроизводимо и гибко преследуют системные цели (пчелиный рой, муравейник, «невидимая рука рынка»). Одной из объективных целей социальной системы в большинстве случаев является сохранение либо увеличение количества агентов и поддержка их функционирования: *самосохранение системы*.

Восприятие, обработка и использование информации социальной системой являются результатом восприятия информации ее агентами, взаимодействий между ними и целеполагания агентов. (Так, формация автоматов, движущихся в заданном направлении по алгоритму «следовать за соседом» и «избегать столкновений», огибает препятствия). Таким образом, социальные системы любого уровня не только преследуют определенные цели, но также, в процессе их достижения, анализируют информацию – как внешнюю (сила стимула или угрозы извне), так и внутреннюю (состояния агентов и их соседей в зоне восприятия). Иными словами, динамика МСС существенно включает *распределенный интеллект*: восприятие, анализ и использование информации социальной системой как единым «актором» и, возможно, как агентом в МСС более высокого уровня.

3. Моделирование интеллекта: состояние проблемы

Мы полагаем, что описание, типология и моделирование социальных систем не могут быть корректными без анализа распределенного интеллекта, которым обладают различные типы МСС. В этом случае особую важность приобретает формальный анализ *индивидуального интеллекта* – прежде всего

человеческого мышления – и степень переносимости его результатов на «коллективный разум».

Анализу умственной деятельности людей и элементов мышления животных посвящена большая область *когнитивных наук* – «междисциплинарных исследований познания, понимаемого как совокупность процессов приобретения, хранения, преобразования и использования знаний живыми и искусственными системами» [18, 19]. В этой области получен огромный фактический материал по психологическим механизмам мышления, структуре сознания и нейрофизиологическим процессам («коррелятам»), сопровождающим работу мозга у человека и животных. К когнитивным наукам примыкает область конструирования и исследования *искусственного интеллекта* [17, 20]. Одним из наиболее известных достижением здесь являются искусственные нейронные сети (ИНС), успешно используемые в аппроксимации функций, распознавании образов, финансовом прогнозировании и поиске многопараметровых корреляций. Важное место в исследованиях ИНС, до настоящего времени остающихся перспективным, но не вполне объясненным техническим средством, занимают изучение механизма их функционирования и осуществляемой ими переработки информации [21].

Несмотря на многочисленные положительные результаты когнитивных исследований и компьютерного моделирования познавательных процессов, общепринятой формальной модели интеллектуальной деятельности до настоящего времени также не существует. В частности, детально разработанный алгоритм «обучения» нейронных сетей позволяет ИНС распознавать лишь такие виды внешнего воздействия, которые были заданы в ходе обучения. За пределами теории остается главное содержание мышления: производство новой, ранее не известной информации.

В то же время в психологии мышления имеется общепринятая описательная схема умственной деятельности, приводящей к открытию новых знаний [22], и выработаны эмпирические рекомендации («эвристики»), повышающие вероятность ее успеха – такие, как «мозговой штурм» [23]. Рядом авторов – в частности, в работах Д.С.Чернавского [24] – был предложен «генетический» механизм отбора *полезной информации*, который закрепляет новое знание, генерируемое в случайном процессе. В исследованиях динамики головного мозга методами электроэнцефалографии (ЭЭГ) и функциональной магнитно-резонансной томографии (фМРТ) экспериментально установлены локализация информации в определенных участках коры [25], «хранение» человеческих знаний и эмоций в различных областях мозга [26].

Характерными особенностями творческой умственной деятельности, установленными в психологии мышления [22, 27], являются:

1. обучение как необходимое условие мышления; желательность возможно более широкой «базы знаний» для плодотворного мышления;

2. спонтанное возникновение правильного решения («озарение», или *инсайт*) у людей и животных, поставленных в «проблемную ситуацию»;
3. повышение вероятности инсайта при активизации случайного перебора вариантов ответа и ослаблении ограничений на варьирование компонентов «проблемной ситуации».

Перечисленные положения использует, в частности, эвристическая схема стимулирования изобретательской деятельности (ТРИЗ). В ее рамках проблемная ситуация названа «изобретательской ситуацией», а изобретение (инсайт) состоит во «взгляде с новой стороны» на компоненты проблемной ситуации и в изменении логической связи между ними [23].

4. Модульная модель интеллекта

Термин «модульность» (modularity) весьма распространен в описании структуры интеллекта и его функций (познания), однако в когнитивных науках он используется скорее на вербальном уровне [28]. Мы предлагаем «модульную» модель индивидуального интеллекта, которая воспроизводит выработку новой информации в процессе мышления и может быть распространена на описание распределенного интеллекта МСС. Наша модель не использует математических соотношений теории информации, ограничиваясь «наивным» представлением об информации как имеющемся (или возникающем) человеческом знании. Ее основой служат перечисленные выше характеристики интеллектуальной творческой деятельности (п.п. 1 – 3) и «угадывание» (а не строгий логический вывод) правильного решения в проблемной ситуации.

Суть модели заключается в блочном (модульном) характере схематичного «опосредованного отражения», или *внутренней репрезентации* [29], вырабатываемой сознанием в ответ на внешнее воздействие, и в генетическом алгоритме подбора модулей, составляющих репрезентацию. Мы предполагаем, что внешнее воздействие на сознание вызывает в нем «отпечаток», интерпретируемый (т.е. воспринимаемый) путем сравнения с некоторой схемой, ранее выработанной для сходных «отпечатков» и вызываемой из памяти. Предполагается, что эта схема (далее – *«образ-схема»*) состоит из небольшого числа знаковых модулей-«иероглифов», библиотека которых составляется в процессе обучения и хранится в памяти (рис. 1). Разные «отпечатки» интерпретируются разными комбинациями модулей. «Единицами хранения» в памяти являются не образы и не аппроксимирующие их схемы: в памяти хранится библиотека модулей и содержатся «ключи» (наборы идентификаторов модулей), позволяющие вызвать ранее построенные из них схемы (рис. 2).

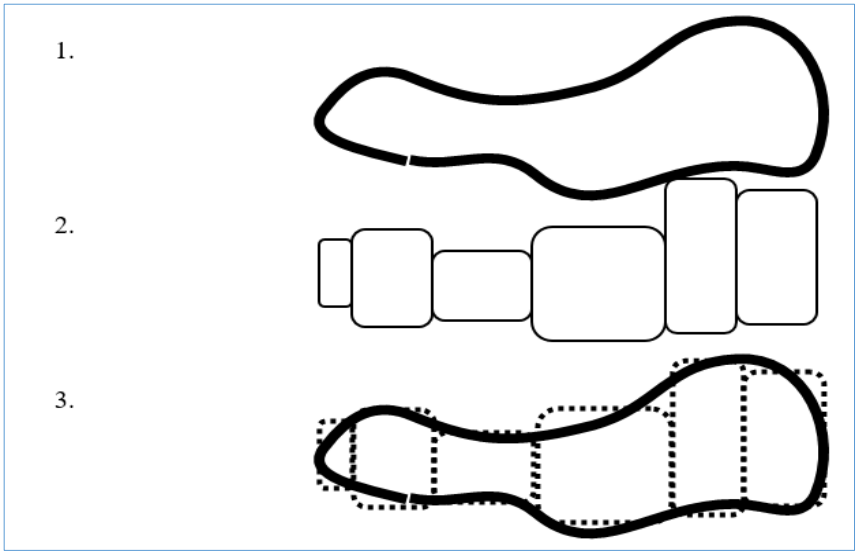


Рис. 1. Компоненты восприятия сознанием внешних стимулов: (1) «отпечаток» внешнего воздействия, (2) его интерпретация («образ- схема»), (3) сопоставление «отпечатка» и «образа- схемы»

Fig. 1. Components of perception of external stimulus: (1) «imprint» of the external action, (2) its interpretation («image-scheme»), (3) comparison of the «image-scheme» with the «imprint»

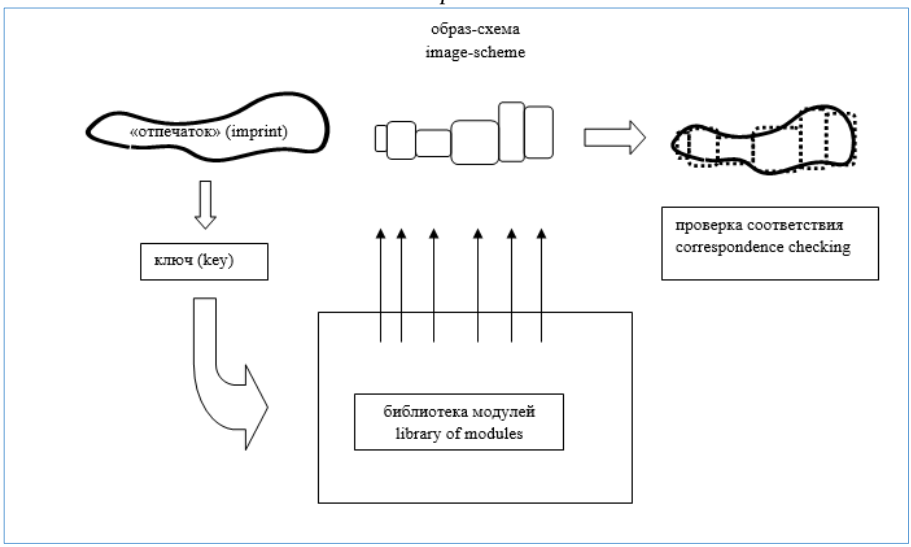


Рис. 2. Алгоритм восприятия: построение «образа-схемы» из библиотеки модулей

Fig. 2. Building of «image-scheme» from the library of modules

По аналогии со знаками иероглифической письменности, модули, составляющие схему, в процессе их приобретения (обучения), вероятно, характеризуют определенные признаки внешних воздействий, но используются далее как универсальные компоненты «образов-схем»: комбинаций модулей, индивидуальных для каждого «отпечатка». Соответствие вызванной модульной схемы «отпечатку» внешнего воздействия проверяется в мозге по некоторому быстрому «автоматическому», не вполне установленному механизму [26].

При хорошем соответствии происходит узнавание. Плохое соответствие вызывает неудовлетворенность (в терминах ТРИЗ – «обостренное противоречие»). Стремление преодолеть неудовлетворенность стимулирует построение новой схемы, лучше соответствующей «отпечатку», из блоков, имеющихся в библиотеке. Создание новой модульной схемы и запоминание ее ключа соответствует производству новой информации в ходе умственной деятельности.

Число модулей, составляющих «образ-схему» (m), можно оценить сверху числом параметров или факторов, которые человек в состоянии воспринимать параллельно («7±2»). Разумной оценкой общего количества N модулей в библиотеке (безусловно, определяемого продолжительностью и глубиной обучения) может служить количество иероглифов, которыми оперирует грамотный носитель японского или китайского языка (до нескольких тысяч). Оценки m~10 и N~1000 дают практически бесконечное число возможных сочетаний модулей в схеме $\binom{m}{N}$~1023. Вместе с тем идентификатор схемы, позволяющий вызвать ее модули из памяти в заданном порядке, может весьма экономно состоять из «номеров» или иных признаков модулей в библиотеке – для чего в предельном случае достаточно одного нейрона, локализованного в коре головного мозга, на один модуль.

Предлагаемая модель на качественном уровне отражает главное содержание интеллектуальной деятельности: «придумывание нового» в форме подбора новой комбинации модулей, которая аппроксимирует воспринимаемый образ лучше всех комбинаций, хранящихся в памяти. Из невозможности, при принятых оценках чисел m и N, построить образ-схему прямым перебором модулей следует достаточно правдоподобный алгоритм творческой интеллектуальной деятельности:

1) поиск наилучшего образа-схемы для «отпечатка» внешнего воздействия по имеющимся в памяти ключам;
2) выявление в найденной схеме модуля (или модулей) с наихудшим соответствием «отпечатку»;
3) замена «плохого» модуля на другие модули, хранящиеся в памяти, ограниченным случайным перебором;
4) нахождение модуля, улучшающего соответствие схемы «отпечатку»;
5) запоминание новой комбинации модулей: введение в память ее ключа (рис. 3).

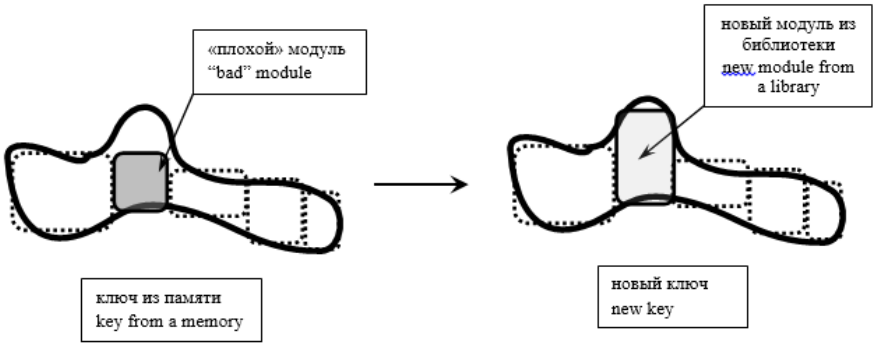


Рис. 3. Замена модуля в образе-схеме, улучшающая соответствие «отпечатку»
Fig. 3. Replacement of a module in the «image-scheme» improving its correspondence to “imprint”

Наша модель согласуется с такими известными условиями успешного решения творческих задач, как предварительное обучение, эрудиция (создание и расширение библиотеки), опыт распознавания проблемных ситуаций (алгоритм нахождения наилучшего образа в памяти), свободный ассоциативный поиск («ключи») и *умственные способности*, т.е. способность без ошибок обрабатывать большие объемы информации. В эмпирических описаниях творческой деятельности пункту (2) нашего алгоритма соответствует «проблемная», или «изобретательская» ситуация, п. (3) – состояние «нерешенной проблемы», которому сопутствуют процессы «расшатывания» образа (и снятие ограничений в поиске как одно из условий успеха), п. (4) – «озарение» (инсайт), а п. (5) – рождение новой информации. В бесструктурной схеме Д.С.Чернавского [24] аналогом перебора модулей является хаотический «перемешивающий слой», или странный аттрактор, в фазовом пространстве континуальной математической модели, а ценность новой информации, полученной случайным путем, определяется ее пригодностью для решения текущих задач. В нашей модели генетический подбор модулей контролируется сопоставлением получаемых схем с «отпечатком» воздействия; хорошее согласие предположительно вызывает биохимически контролируемое удовлетворение [26, 30].

Обсуждаемая модель согласуется с наличием локализованных нейронов, воспроизводимо активируемых в коре головного мозга при предъявлении определенного стимула («когнитивной специализацией нейронов в мозге человека» [25]). Локализованные нейроны в этом случае являются частью физического «субстрата» модуля, или воспроизводимо активируемого ансамбля нейронов [31], в составе «образа-схемы», возникающей при распознавании внешнего воздействия. В терминах, используемых при

обсуждении сетевых структур мозга, физическим носителем модуля является *ког*: «распределенная группа нейронов, сцепленная единым когнитивным опытом», т.е. кодирующая определенный аспект восприятия внешних воздействий [25]. Следует заметить, что нейроны, входящие в состав «паттерна активации» [31], могут быть локализованы в коре головного мозга без определенного алгоритма, ситуативно – подобно записи информации на магнитный диск. В этом случае тесная связь мышления с конкретными сетевыми структурами мозга (*коннектомом*) и сознания (*когнитомом*), постулируемая в данной области [25], подобно предполагаемой связи сетевой структуры с динамикой социальных систем, может не соответствовать действительности (см. разд. 1).

5. К моделированию распределенного интеллекта

Предлагаемая «модульная» модель индивидуального интеллекта естественно распространяется на описание распределенного интеллекта (РИ) мультиагентных социальных систем. Наличие интеллекта у МСС определяется способностью агентов воспринимать информацию, наличием целей у агентов и их взаимодействием (см. Введение). Уровень РИ соответственно зависит от когнитивных возможностей агентов, от преследуемых ими целей, от характера и структуры межагентных взаимодействий, а также от внешних условий.

При распаде структуры МСС (отсутствие ресурсов, изобилие ресурсов либо «война всех против всех») коллективная обработка информации не реализуется. Распределенный интеллект системы в общем случае возрастает с увеличением числа агентов и с усложнением структуры их взаимодействий.

Анализ и использование информации организационными системами достаточно высокого уровня, функционирующими в человеческом обществе, имеет очевидные аналогии с описанием индивидуального интеллекта по «модульной» схеме (табл. 1). В качестве «отпечатка» внешнего воздействия на систему (например, аварии на производстве) здесь выступает гетерогенная совокупность реакций системы как целого и индивидуальных реакций агентов. (Так, по должностным инструкциям, работники предприятия при аварии обязаны сообщить о ней руководству, однако серьезная авария может вызвать панику и потерю управления).

Составными единицами при интерпретации воздействия системой служат инструкции и должностные обязанности. Сильные внешние воздействия изменяют структуру системы (при аварии – эвакуация персонала, создание аварийного штаба и комиссии). «Образом-схемой» в данном случае является стандартный ответ системы (действие по инструкциям), корректировкой схемы – оценка успешности действий и испытание новых предложений (работа комиссии), заменой модуля (модулей) в «образе-схеме» – выбор оптимальных действий (выводы комиссии), запоминанием новой информации – дополнение и изменение инструкций.

Табл. 1. Сравнение компонентов индивидуальной интеллектуальной деятельности и распределенного интеллекта организационной системы
Table 1. A comparison of activity of individual human's intellect with a distributed intelligence of an enterprise

индивидуальный интеллект человека	распределенный интеллект предприятия
1-й этап восприятия: «отпечаток» внешнего воздействия	сумма системных и индивидуальных реакций работников на внешнее воздействие
библиотека модулей и «хранилище ключей»	рабочая информация, должностные инструкции
2-й этап восприятия: выбор модульной «схемы-образа»	оценки параметров воздействия и его предварительная характеристика; действия по инструкциям
поиск лучшей имеющейся приближенной схемы, оценка ее расхождений с образом	оценка успешности стандартных действий («создание комиссии»)
случайная замена модулей	корректировка действий; испытание новых предложений («работа комиссии»)
«генетический контроль»	оценка успешности скорректированных действий («выводы комиссии»)
запоминание новой схемы	дополнение и изменение инструкций

Представленный модельный пример иллюстрирует как сложность и неоднородность структурированной социальной системы, так и наличие нескольких уровней ее распределенного интеллекта (так, при потере управления персонал предприятия будет действовать группами, преследующими собственные цели). Существенно, что рациональность системной динамики увеличивается при ограничении набора возможных действий агентов и при наличии «библиотеки» стандартных реакций руководства предприятия и персонала.

Тесную связь «интеллектуальности» организационной системы с формализацией ее структуры и действий иллюстрирует табл. 2. Рациональность индивидуального поведения людей также возрастает с сокращением возможностей выбора (переход улицы по светофору, покупка или продажа акций на бирже по фиксированной цене) и становится «ограниченной» при выборе из многих возможностей (переход улицы без светофора, произвольная цена акций на бирже).

Табл. 2. Формальные условия функционирования разных организационных систем
Table 2. Formal conditions of activity for different organizing systems

«идеальная комиссия»	митинг	«коллективный идиот»
общие знания по специальности	общие намерения	ничего общего
целевой отбор участников по критерию квалификации	случайный отбор участников по близости	свободный вход

	настроений	
сильное управление (председатель с решающим голосом)	слабое управление	нет управления
формализованный обмен информацией и мнениями, исключение эмоций	неформальный обмен мнениями и эмоциями	случайный обмен эмоциями
количественное сравнение значимости мнений (голосование)	декларации мнений (призывы)	нет формулируемых мнений
подчинение меньшинства большинству	неподчинение меньшинства большинству	нет большинства
обязательность исполнения решений	необязательность исполнения решений	нет решений

Общеизвестные схемы коллективного поведения «живых» социальных систем с более примитивными агентами (улей, муравейник, гнездо ос) также неплохо согласуются с основными этапами «модульного» алгоритма интеллектуальной деятельности [16]. Как и в организационной системе, состоящей из людей, в поведении насекомых трудно разделить формирование «отпечатка» (фиксацию системой внешнего воздействия) и стандартную реакцию на него («образ-схему»); примером может служить атака муравьев или ос на чужеродный объект вблизи гнезда. (Разделение воздействия, его отражения в сознании и репрезентаций образа в психологии также является непростой задачей [19]). «Танцы» пчел в улье и «тропинки» муравьев, которые можно рассматривать как модули, направляющие движения особей, изменяются при изменении внешних условий. Перемещение муравьиного гнезда из неблагоприятной зоны в благоприятную, восстановление поврежденного муравейника и многие другие действия коллективных насекомых разбиваются на несколько стандартных воспроизводимых режимов (возбуждение, несогласованные действия, согласованные действия, «успокоение» при достигнутой цели) – этот перечень легко продолжить.

6. «Измерение» распределенного интеллекта модельной системы

Компьютерное моделирование динамики МСС неявно включает оценку уровня распределенного интеллекта модельной системы (например, транспортного потока [11] – по скорости движения, пропускной способности дороги, предсказываемому числу аварий и т. д.) и максимизацию этого уровня. Мерой РИ в таких расчетах является близость достигаемого результата к объективной цели системы – в случае транспортного потока к максимально быстрому безаварийному движению. Для компьютерной иллюстрации этих положений мы использовали анимационную модель, в которой агенты с одинаковым радиусом r_0 перемещались по коридору с препятствиями, блокирующими сквозное движение (рис. 4). Состояние агента

задавали пять числовых параметров (R(n), k_1, k_2, k_3, k_4), где R = nr_0 – радиус восприятия агентом его окружения, а остальные $0 \le k_i \le 0.1$ – параметры движения агента:

k_1 – ускорение вправо: приращение скорости на каждом шаге дискретного времени как доля максимальной скорости v_0 (*целеполагание* агента),

k_2 – торможение перед препятствием в радиусе восприятия,

k_4 и k_3 – соответственно следование за другими агентами в радиусе восприятия и коррекция собственной скорости по скорости их движения.

Столкновение агента с препятствиями и другими агентами происходили как абсолютно упругие с соответствующими изменениями направления движения и скоростей. Более детальное описание схемы расчета будет представлено в отдельной публикации.

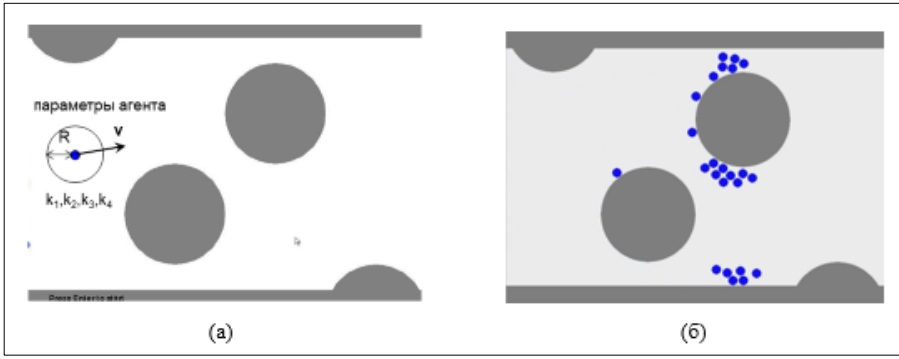


Рис.4. (а) Схема «коридора» с препятствиями и параметры агента, (б) движение в «обученной» системе
Fig. 4. (a) Pathway with obstacles and agent's parameters; (b) movement of agents in a «trained» system

В первом цикле расчета 24 агента одинакового радиуса r0 с фиксированным радиусом восприятия R=3r0 и параметрами движения (0.01, 0, 0, 0), случайным образом размещенные в полосе шириной 5r0 перед «стартовой» линией, перемещались по коридору слева направо до «финиша», т.е. прохождения полной длины коридора. Направления движения и скорости агентов корректировались на каждом шаге дискретного времени в соответствии с параметрами {ki} по объектам (препятствиям и другим агентам) в радиусе восприятия. В последующих циклах параметры движения 12 «медленных» агентов, преодолевших коридор на предыдущем цикле позже 12 «быстрых», заменяли параметрами 12 «быстрых» агентов со случайными вариациями {□ki=±0.001} (генетический отбор). Количественной оценкой РИ системы служило среднее время t прохождения коридора агентом в каждом цикле.

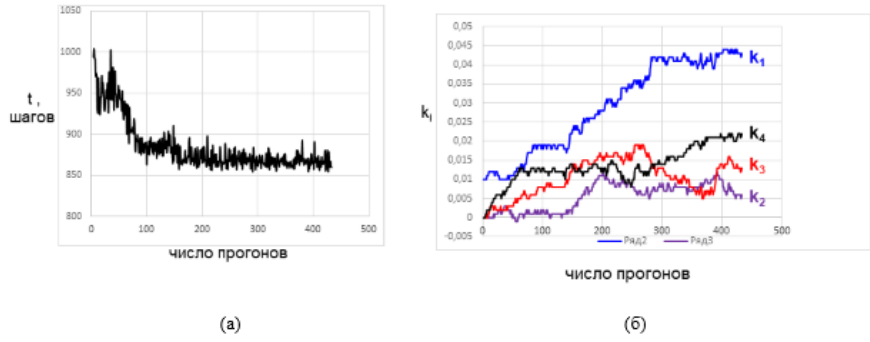


Рис.5. (а) Изменение среднего времени прохода ***t*** *при обучении» системы, (б) изменения коэффициентов движения*

Fig. 5. (a) Evolution of average passing time t during 'learning' (b) evolution of movement parameters

В ходе эволюции искусственной МСС среднее время прохода t за первые 150-200 циклов уменьшалось от 1040–1050 до 880–870 шагов и далее оставалось приблизительно постоянным (рис. 5 а). Параметры движения агентов при этом продолжали изменяться (рис. 5 б). Характер движения в результате эволюции изменялся от хаотического дрейфа при многочисленных столкновениях с препятствиями и между собой в «необученной» системе к согласованному движению группами, огибающими препятствия, в «обученной» системе (рис. 4 б). Время прохождения коридора агентами с фиксированными параметрами движения (k1, 0.01, 0.01, 0.01) при $k1 \leq 0.05$ уменьшалось с увеличением радиуса восприятия, проходя через минимум (рис. 6). (Дальнейшее увеличение k1 приводило к достижению всеми агентами максимальной скорости v0 за первые 10-15 шагов дискретного времени и сопровождалось увеличением t с его небольшим линейным ростом при увеличении R). Таким образом, расчеты наглядно продемонстрировали способность модельной МСС из искусственных агентов воспринимать информацию и оптимизировать динамику в ходе «обучения» (т.е. генетического отбора «быстрых» агентов).

7. Обсуждение

Проявлениям РИ в социальных системах, включая различные аспекты коллективной человеческой деятельности, посвящена обширная литература (см. [1, 10, 30, 32, 33]). Однако в большинстве современных публикаций интеллектуальная деятельность МСС трактуется весьма ограниченно: как обмен информацией между агентами [1, 33] или как выработка общего мнения

в совокупности людей («crowd wisdom» [34]). За пределами обсуждения, аналогично теориям индивидуального интеллекта, оказывается способность МСС не только воспринимать и обрабатывать информацию, но и генерировать новую информацию – например, в форме муравьиных «тропинок», ведущих к новому, не использованному ранее источнику пищи. При этом «обучаемая» система может состоять из агентов с нулевыми когнитивными возможностями – как в построенной нами модели движения по коридору, где динамику каждого агента задавали пять числовых параметров.

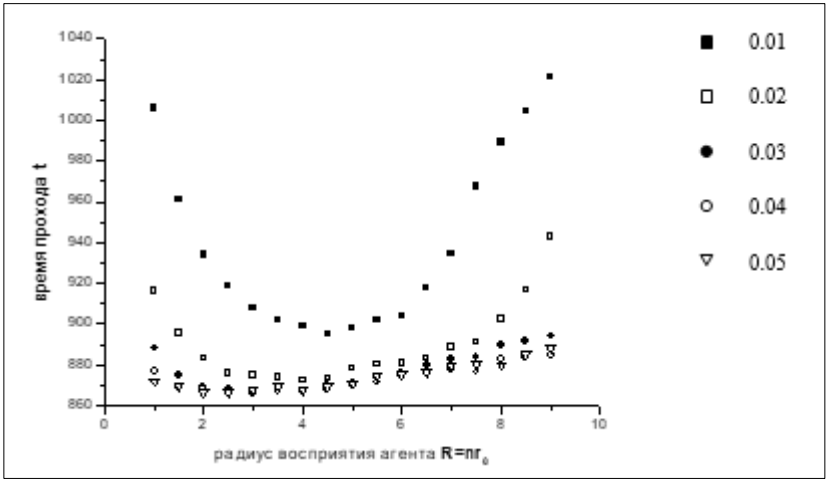


*Рис.6. Зависимость времени прохода **t** от радиуса восприятия **R** при разных коэффициентах $\mathbf{k_1}$*

*Fig. 6. Passing time **t** vs. perception radius **R** at different values of movement parameter $\mathbf{k_1}$*

Организационные системы, состоящие из людей, также способны не только к обмену индивидуальными мнениями, приводящему к «консенсусу» (что воспроизводится большинством социологических моделей, см. [14, 15]), но и к производству новой информации «коллективным разумом». РИ таких систем отличается от человеческого сознания как принципиально меньшим числом «узлов» (т.е. людей) в сетевых структурах, так и способностью этих узлов к глубокой переработке информации. Текущая информация о состоянии, генерируемая в системе, направляет ее коллективную динамику в переменных внешних условиях и фиксируется в виде административных «образов-схем» – т.е. распоряжений, правил и инструкций. При этом «глубина» РИ явно возрастает с повышением формализации связей в структуре МСС. Так, в модельных примерах таблицы 2, строгая регламентация «идеальной комиссии» (левая колонка) является эвристикой, стимулирующей ее содержательную работу, тогда как полностью деформализованная МСС (правая колонка таблицы), независимо от интеллектуального уровня

составляющих ее людей, вряд ли способна принять какое-либо общее решение.

«Модульный» подход представляется продуктивным также для конструирования искусственного интеллекта (ИИ) в вычислительной компьютерной среде. Известно, что при «обучении» многослойных нейронных сетей векторные реплики распознаваемых объектов в промежуточных слоях ИНС преобразуются к модульной структуре [21]. В общем случае распознаваемому объекту может сопоставляться набор модулей, фиксирующих определенные качества объекта – что открывает путь к созданию самообучающихся программ. В отличие от индивидуального человеческого интеллекта, число модулей в «образе-схеме», при хороших вычислительных мощностях, может быть достаточно большим, а перебор «плохих» модулей – быстрым и эффективным. Параметры «модульного» ИИ, таким образом, могут существенно превысить любые человеческие возможности, что указывает на вероятные риски при его реализации.

8. Перспективные направления исследований

Исследования распределенного интеллекта социальных систем будут проводиться в нескольких основных направлениях.

(1) Разработка модульной модели индивидуального интеллекта, пока сформулированной на качественном уровне, наполнение ее количественным содержанием (структура модулей, ключей и библиотек, модульный состав «образов-схем», механизмы сравнения «отпечатка» с модульной схемой, замены модулей в схеме, обновления модулей в библиотеке и т.д.).

(2) Эмпирические оценки уровня распределенного интеллекта для социальных систем разной природы (организаций, партий, политических течений и т.д.). Поиск корреляций «уровня интеллекта» со структурой социальной системы.

(3) Формальное описание процессов восприятия и переработки информации индивидуальным сознанием и «коллективным разумом». Исследование математических свойств *информационного пространства*, объектами которого служат модули восприятия и «образы-схемы».

(4) Модификация существующих моделей коллективного поведения (динамика пешеходных и автомобильных потоков, распространение мнений в онлайновых социальных сетях, политические кампании и др.) с непосредственным учетом распределенного интеллекта у системы взаимодействующих агентов. Сопоставление результатов моделирования с эмпирическими данными.

9. Заключение

Мы полагаем, что предложенный нами «модульный» алгоритм восприятия, обработки, использования и хранения информации применим к описанию индивидуального интеллекта человека и (в скорректированном виде) к моделированию распределенного интеллекта социальных систем. Оба вида информационной динамики реализуются в определенных сетевых структурах (см. [1, 8, 14, 17, 25]), взаимосвязь которых с уровнем интеллекта следует дополнительно изучать. Очевидными направлениями дальнейших исследований в данной области являются детализация и формализация модульной модели интеллекта, эмпирические оценки «глубины» распределенного интеллекта МСС и его непосредственный учет в моделировании мультиагентных систем.

Ю.Л.С. признателен доктору психологических наук М.В. Фаликман (НИУ ВШЭ) за предоставленную литературу по когнитивным исследованиям и плодотворное обсуждение «модульной» модели.

Список литературы

[1] Moussaïd M, Helbing D, Theraulaz G, How simple rules determine pedestrian behavior and crowd disasters. PNAS, 108 (17), 2011, pp. 6884-6888
[2] Губко М.В., Новиков Д.А. Теория игр в управлении организационными системами. 2-е издание. М.: Синтег, 2005, 138 с.
[3] Galam S. Sociophysics: a physicist's modeling of phycho-polytical phenomena. Springer, 2012, 439 p.
[4] Захаров А.В. Модели политической конкуренции: обзор литературы. Экономика и математические методы, том 45, вып. 1, 2009 г., стр. 110-128
[5] Dorogovtsev S.N. Lectures on Complex Networks. Clarendon: Oxford, 2010, 134 p.
[6] Newman M.E.J. The structure and functions of complex networks, SIAM Review, 45(2), 2003, pp. 167-225.
[7] Берновский М.М., Кузюрин Н.Н. Случайные графы, модели и генераторы безмасштабных графов. Труды ИСП РАН, 2012, том 22, стр. 419-432. DOI: 10.15514/ISPRAS-2012-22-22.
[8] Евин И.А. Введение в теорию сложных сетей. Компьютерные исследования и моделирование, том 2, вып. 2, 2010 г., стр. 121-141.
[9] Новиков Д.А. Модели стратегической рефлексии. Автоматика и телемеханика, том 73, вып.1, 2012 г., стр. 1 -19
[10] Kahneman D. Maps of Bounded Rationality: Psychology for Behavioral Economics Amer. Econ. Rev., 2003, 93(5), pp. 1449-1475.
[11] Гасников А.В. (ред.), Введение в математическое моделирование транспортных потоков. М.: МЦНМО, 2013, 428 с.
[12] Адамчук А.Н., Есипов С.Е. Коллективно флуктуирующие активы при иналичии арбитражных возможностей и оценка платежных обязательств. Усп. физ. наук, том 167, вып. 12, 1997 г., стр. 1295-1306.
[13] Schelling T. Dynamic models of segregation. J. Math. Sociol., 1 (2), 1971, pp. 143-186
[14] Castellano C., Fortunato S., Loreto V. Statistical physics of social dynamics. Rev. Mod. Phys. 81 (2), 2009, pp. 591-646.

[15] Словохотов Ю.Л. Физика и социофизика. Проблемы управления, 2012, вып. 3, стр. 2–34.
[16] Кипятков В.Е. Мир общественных насекомых, 3-е изд., М.: Либроком, 2009, 408 с.
[17] Engelbtecht A.P. Fundamentals of computational swarm intelligence. N.-Y.: Wiley, 2005, 672 p.
[18] Фаликман М.В. Основные подходы в когнитивной науке. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2017-02-09/presentation.pdf. Дата обращения: 02.04.2018.
[19] Величковский Б.М. Когнитивная наука: основы психологии познания. В 2-х т.. М.: Смысл; Академия, 2006.
[20] Shaib-Draa B, Moulin B., Mandiau P., Millot P. Trends in distributed artificial intelligence. Artific. Intelligence Rev., 6 (1), 1992, pp. 35-66.
[21] Хайкин С. Нейронные сети. Полный курс. 2-е издание. Вильямс, 2016, 1104 с.
[22] Петухов В.В. Психология мышления. Учебно-методическое пособие. М.: МГУ, 1987, 99 с.
[23] Альтшуллер Г.С. Найти идею. Введение в ТРИЗ – теорию решения изобретательских задач. 4-е изд. М.: Альпина Паблишерз, 2011, 400 с.
[24] Чернавский Д.С. Синергетика и информация: динамическая теория информации. 3-е изд. М.: Либроком, 2009, 304 с.
[25] Анохин К.В. Когнитом: разум как физическая и математическая структура. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2016-09-27/presentation.pdf. Дата обращения: 02.04.2018.
[26] Шумский С.А. Моделирование работы мозга: состояние и перспективы. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2015-03-31/presentation.pdf. Дата обращения: 02.04.2018.
[27] Ohlsson S. Information-processing explanations of insight and related phenomena. In M. T. Keane & K. J. Gilhooly (Eds.), Advances in the psychology of thinking. New York, NY: Harvester Wheatsheaf, 1992, pp. 1– 44.
[28] Fodor J.A. The Modularity of Mind. MIT Press 1983,142 p.
[29] Курганский А.В. Понятие внутренней репрезентации в когнитивной нейронауке. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2017-02-28/presentation.pdf. Дата обращения: 02.04.2018.
[30] Dandurand F., Shultz T.R., Rivest F. Complex problem solving with reinforcement learning. Proc. 6th IEEE Internat. Conf. on Development and Learning, 2007, pp. 157-162.
[31] Hebb D.O. The Organization of Behavior: a Neuropsychological Theory, Wiley, 1949, 335 p.
[32] Mossaid M., Garnieer S., Theraulaz G., Helbing D. Collective information processing and pattern formation in swarms, flocks and crowds. Topics Cogn. Sci., 1, 2009, pp 469-497.
[33] Becker J., Brackbill D., Centola D. PNAS, 114, 2017, pp. E5070-E5076.

Toward construction of a modular model of distributed intelligence

[1,2]*Yu.L. Slovokhotov <slov@phys.chem.msu.ru>*
[3]*I.S. Neretin <ivan@neretin.ru>*
[1]*Lomonosov Moscow State University,*
GSP-1, 1 Leninskie Gory, Moscow, 119991, Russia
[2]*Institute of Organoelement Compounds, Russian Academy of Sciences,*
28 Vavilov St., Moscow, 119991, Russia
[3]*Rock Flow Dynamics, 25A Profsoyuznaya St., Moscow, 117418, Russia*

Abstract. Multi-agent social systems (MASS) in general are systems of autonomous interdependent agents each pursuing its own goals interacting with other agents and environment. Dynamics of MASS cannot be adequately modeled by the methods borrowed from statistical physics since these methods do not reflect the main feature of social systems, viz. their ability to percept, process and use the external information. This important quality of distributed ("swarm") intelligence has to be directly taken into account in a correct theoretical description of social systems. However, discussion of distributed intelligence (DI) in the literature is mostly restricted to distributed tasks, information exchange and aggregated judgment – i.e. to 'sum' or 'average' of independent intellectual activities. This approach ignores empirically well-known option of a 'collective insight' in a group as a special demonstration of MASS's DI . It this paper, a state of art in modeling social systems and studies of intelligence per se are briefly characterized, and a new modular model of intelligence is suggested. The model allows to reproduce the most important result of intellectual activity, i.e. creation of new information, which is not reflected in the contemporary theoretical schemes (e.g. neural networks). Using the "modular" approach, a correspondence between individual intelligence and DI of MASS is discussed, and prospective directions for future studies are suggested. Efficiency of DI was estimated numerically by computer simulations of a simple system of agents with variable kinematic parameters {ki}, moving through a pathway with obstacles. Selection of fast agents with 'positive mutation' of parameters gives ca. 20% reduction of average passing time after 200-300 cycles and creates a swarm movement where agents follow a leader and cooperatively avoid obstacles.

Keywords: multi-agent social systems; swarm intelligence; models of intelligence.

DOI: 10.15514/ISPRAS-2018-30(3)-23

For citation: Slovokhotov Yu.L., Neretin I.S. Toward construction of a modular model of distributed intelligence. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 3, 2018, pp. 341-362 (in Russian). DOI: 10.15514/ISPRAS-2018-30(3)-23

References

[1] Moussaïd M, Helbing D, Theraulaz G, How simple rules determine pedestrian behavior and crowd disasters PNAS, 108 (17), 2011, pp. 6884-6888
[2] Goubko M.V.,Novikov D.A. Game theory in control of organizing systemsT. 2-nd ed.. Moscow, Sinteg, 2005, 138 p. (in Russian)

[3] Galam S. Sociophysics: a physicist's modeling of phycho-polytical phenomena. Springer, 2012, 439 p.
[4] Zakharov A.V. Models of political competition^ a review. Economics and Mathematical Methods [Ekonomika I matematicheskiye metody], 45 (1), , 2009 pp. 110-128 (in Russian)
[5] Dorogovtsev S.N. Lectures on Complex Networks. Clarendon: Oxford, 2010, 134 p.
[6] Newman M.E.J., The structure and functions of complex networks, SIAM Review, 45(2), 2003, pp. 167-225.
[7] Bernovskiy M.M., Kuzyurin N.N. Random graphs, models and generators of scale-free graphs. Trudy ISP RAN/Proc. ISP RAS, 2012, v. 22, pp. 419-432 (in Russian). DOI: 10.15514/ISPRAS-2012-22-22.
[8] Yevin I.A. Introduction to a theory of complex networks. Computer Research and Modeling [Kompyuternye issledovaniya I modelirovanie], 2 (2), 2010, pp. 121-141 (in Russian).
[9] Novikov D.A. Models of strategic behavior. Automation and Remote Control, 73 (1), 2012, pp. 1 -19
[10] Kahneman D. Maps of Bounded Rationality: Psychology for Behavioral Economics Amer. Econ. Rev., 2003, 93(5), pp. 1449-1475.
[11] Gasnikov A.V. (Ed.). Introduction into mathematic modeling of traffic flows. Moscow, MTsIMO, 2013, 428 p. (in Russian)
[12] Adamchuk A.N., Esipov C.E. Collectively fluctuating assets in the presence of arbitrage opportunities, and option pricing. Phys. Usp., vol. 40, 1997, pp. 1239–1248
[13] Schelling T. Dynamic models of segregation. J. Math. Sociol., 1 (2), 1971, pp. 143-186
[14] Castellano C., Fortunato S., Loreto V. Statistical physics of social dynamics. Rev. Mod. Phys. 81 (2), 2009, pp. 591-646.
[15] Slovokhotov Yu.L. Physics vs. Sociophysics. Control Science [Problemy upravleniya], 2012 (3), pp. 2-34 (in Russian).
[16] Kipyatkov V.E. World of social insects, 3rd Ed., Moscow, Librokom, 2009, 308 p.
[17] Engelbtecht A.P. Fundamentals of computational swarm intelligence. N.-Y.: Wiley, 2005, 672 p.
[18] Falikman M.V. The principal approaches in cognitive science http://www.soc-phys.chem.msu.ru/rus/prev/zas-2017-02-09/presentation.pdf (in Russian). Accessed 02.04.2018.
[19] Velichkovsky B.M. Cognitive science: foundation of cognition psychology. In 2 volumes.Moscow, Smysl, Akademiya, 2006 (in Russian).
[20] Shaib-Draa B, Moulin B., Mandiau P., Millot P. Trends in distributed artificial intelligence. Artific. Intelligence Rev., 6 (1), 1992, pp. 35-66.
[21] Khaikin S. Neural networks. A comprehensive course. 2nd edition. Viljams, 2016, 1104 p. (in Russian)
[22] Petukhov V.V. Psykhology of thinking A textbook. Moscow, MGU,. 1987, 99 p. (in Russian).
[23] Altshuler G.S. To find the idea. Introduction into TRIZ: a theory of solving inventional problems. 4EP ed. Moscow, Alpina Publishers, 2011, 400 p. (in Russian)
[24] Chernavsky D.S. Synergetics and information: dynamic theory of information. 3rd ed. Moscow, Librokom, 2009, 304 p. (in Russian)
[25] Anokhin K,V. Cognitom: mind as a physical and mathematical structure. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2016-09-27/presentation.pdf (in Russian). Accessed 02.04.2018.

[26] Shumsky S.A. Modeling of brain activity: state of art and prospects. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2015-03-31/presentation.pdf (in Russian). Accessed 02.04.2018.
[27] Ohlsson S. Information-processing explanations of insight and related phenomena. In M. T. Keane & K. J. Gilhooly (Eds.), Advances in the psychology of thinking, New York, NY: Harvester Wheatsheaf, 1992, pp. 1– 44.
[28] Fodor J.A. The Modularity of Mind. MIT Press 1983,142 p.
[29] Kurgansky A.V. Internal representation in cognitive neuroscience. http://www.soc-phys.chem.msu.ru/rus/prev/zas-2017-02-28/presentation.pdf (in Russian). Accessed 02.04.2018.
[30] Dandurand F., Shultz T.R., Rivest F. Complex problem solving with reinforcement learning Proc. 6th IEEE Internat. Conf. on Development and Learning, 2007, pp 157-162.
[31] Hebb D.O. The Organization of Behavior: a Neuropsychological Theory, Wiley, 1949, 335 p.
[32] Mossaid M., Garnieer S., Theraulaz G., Helbing D. Collective information processing and pattern formation in swarms, flocks and crowds. Topics Cogn. Sci., 1, 2009, pp. 469-497.
[33] Becker J., Brackbill D., Centola D. PNAS, 114, 2017, pp. E5070-E5076.