DOI: 10.15514/ISPRAS-2021-33(4)-13

Полная решающая процедура для теории ограниченной адресной арифметики

[1,2] Р.Ф. Садыков, ORCID: 0000-0002-2792-2465 <sadykov@ispras.ru>
[2] М.У. Мандрыкин, ORCID: 0000-0002-9306-7719 <mandrykin@ispras.ru>
[1] Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1
[2] Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Процесс разработки кода на Си довольно часто сопровождается появлением ошибок, связанных с использованием указателей и адресов памяти. В связи с этим возникает потребность создания инструментов автоматизированной проверки программ. Одним из методов, применяемых такими инструментами проверки, является использование решающих процедур на основе SMT-решателей. Но в то же время разрешимые логики (комбинации логических теорий), требуемые для адекватного моделирования указателей в языке Си, непосредственно не присутствуют в стандарте SMT-LIB и не реализованы в большинстве существующих SMT-решателей. Одним из возможных способов поддержки таких логик является непосредственная их реализация в каком-либо SMT-решателе, однако такой подход часто оказывается трудоемким (требуется изменение исходного кода решателя), негибким (трудно модифицировать сигнатуру и семантику новых теорий) и ограниченным (требуется отдельная реализация поддержки теории в каждом используемом решателе). Другим способом является реализация пользовательских стратегий конечного инстанцирования кванторов для сведения формул в неподдерживаемых логиках к формулам в широко распространенных разрешимых логиках, таких как QF_UFLIA. В данной статье представлена процедура инстанцирования лемм для трансляции формул в теории типизированной адресной арифметики в логику QF_UFLIA. Для процедуры трансляции даны доказательства корректности и полноты, а также описана формализация этих доказательств в системе Isabelle/HOL. Сама теория адресной арифметики сформулирована на основе известных ошибок использования адресной арифметики в языке Си, а также спецификаций семантики этих операций из стандарта языка. Аналогичные доказательства и процедура также могут быть сформулированы для фрагмента теории бит-векторов (монотонные формулы над равенствами между выражениями с побитовыми операциями, такими как исключающее “или”, сдвиг, конкатенация, экстракция, отрицание). Представленная в статье процедура трансляции, в частности, позволяет легко реализовать полный метод доказательства утверждений в теории адресной арифметики в системе Isabelle/HOL на основе существующей в этой системе поддержки SMT-решателей (Z3 или VeriT).

Ключевые слова: статическая верификация; задача выполнимости формул в теориях; адресная арифметика; решающие процедуры

Для цитирования: Садыков Р.Ф., Мандрыкин М.У. Полная решающая процедура для теории ограниченной адресной арифметики. Труды ИСП РАН, том 33, вып. 4, 2021 г., стр. 177-194. DOI: 10.15514/ISPRAS–2021–33(4)–13

Благодарности. Работа выполнена при поддержке Минобрнауки России в рамках проекта №АААА-А19-119110790086-3.

Complete decision procedure for the theory of bounded pointer arithmetic based on quantifier instantiation and SMT

[1,2] R. Sadykov ORCID: 0000-0002-2792-2465 <sadykov@ispras.ru>
[2] M. Mandrykin ORCID: 0000-0002-9306-7719 <mandrykin@ispras.ru>
[1] Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia
[2] Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The process of developing C programs is quite often prone to errors related to the uses of pointer arithmetic and operations on memory addresses. This promotes a need in developing various tools for automated program verification. One of the techniques frequently employed by those tools is invocation of appropriate decision procedures implemented within existing SMT-solvers. But at the same time both the SMT standard and most existing SMT-solvers lack the relevant logics (combinations of logical theories) for directly and precisely modelling the semantics of pointer operations in C. One of the possible ways to support these logics is to implement them in an SMT solver, but this approach can be time-consuming (as requires modifying the solver’s source code), inflexible (introducing any changes to the theory’s signature or semantics can be unreasonably hard) and limited (every solver has to be supported separately). Another way is to design and implement custom quantifier instantiation strategies. These strategies can be then used to translate formulas in the desired theory combinations to formulas in well-supported decidable logics such as QF_UFLIA. In this paper, we present an instantiation procedure for translating formulas in the theory of bounded pointer arithmetic into the QF_UFLIA logic. We formally proved soundness and completeness of our instantiation procedure in Isabelle/HOL. The paper presents an informal description of this proof of the proposed procedure. The theory of bounded pointer arithmetic itself was formulated based on known errors regarding the correct use of pointer arithmetic operations in industrial code as well as the semantics of these operations specified in the C standard. Similar procedure can also be defined for a practically relevant fragment of the theory of bit vectors (monotone propositional combinations of equalities between bitwise expressions). Our approach is sufficient to obtain efficient decision procedures implemented as Isabelle/HOL proof methods for several decidable logical theories used in C program verification by relying on the existing capabilities of well-known SMT solvers, such as Z3 and proof reconstruction capabilities of the Isabelle/HOL proof assistant.

Keywords: static verification; quantifier instantiation; SMT formulas; SMT solvers; automated decision procedures; software verification

For citation: Sadykov R., Mandrykin M. Complete decision procedure for the bounded theory of pointer arithmetic based on quantifier instantiation and SMT. Trudy ISP RAN/Proc. ISP RAS, vol. 33, issue 4, 2021. pp. 177-194 (in Russian). DOI: 10.15514/ISPRAS–2021–33(4)–13

Acknowledgments. This work was supported by the Ministry of Education and Science of the Russian Federation within the framework of project No. AAAA-A19-119110790086-3.

1. Введение

Идея формулирования специфических логических теорий, разрешимых с помощью трансляции в одну из логик, поддерживаемых современными SMT-решателями, возникла у авторов данной статьи в контексте дедуктивной верификации Си-программ в среде Isabelle-HOL. Поэтому вначале рассмотрим проблемы, возникающие при автоматизации рутинных доказательств (часто необходимых при верификации программ) в системах интерактивной верификации, таких как Isabelle/HOL.

На сегодняшний день в области автоматизированной дедуктивной верификации Си-программ распространены два основных подхода к автоматизации доказательств – использование специфических тактик и использование общих решающих процедур, таких как SMT- и суперпозиционные решатели. При этом оба подхода на практике обладают достаточно существенными недостатками. Подход с использованием узко специфических тактик часто страдает из-за слишком узко определенного набора преобразований или

доказательств, выполняемых или поддерживаемых каждой конкретной тактикой, таких как, например, замена некоторого выделенного подтерма на эквивалентный (переписывание) или поддержка теории линейной целочисленной арифметики над неинтерпретируемыми константами. Хотя каждое конкретное применение такой узко специфичной тактики, как правило, приводит к хорошо определенному и вполне ожидаемому результату, число взаимодействий с интерактивным инструментом верификации, необходимое для доказательства типично возникающих целей при использовании специфичных тактик обычно оказывается весьма большим, что сильно увеличивает трудоемкость верификации. Использование же общих решающих процедур обычно характеризуется большой непредсказуемостью результата, так как лежащие в основе таких инструментов подходы в общем случае не обладают полнотой для формул из того класса, для которого они часто применяются. При этом в силу неполноты практическое значение обычно имеет только успешное применение решателя для полного разрешения какой-либо цели доказательства, возможность которого часто трудно предсказать заранее и которое часто оказывается неустойчивым к небольшим изменениям целевого утверждения.

Некоторым продвижением по сравнению с описанными практически используемыми подходами кажется применяемый в инструменте Isabelle/HOL [1] подход на основе структурированных доказательств на языке Isabelle/Isar, в котором каждая цель доказательства явно ставится пользователем с учетом возможностей реализованных в инструменте методов автоматизации логического вывода, что позволяет наилучшим образом подстраивать структуру доказательств под возможности инструментов автоматического доказательства. В сочетании с достаточно мощными методами, основанными на переписывании термов (*simp*, *auto*), методе табло (*blast*) и использовании суперпозиционных (*metis*, *meson*) и SMT-решателей (*smt*), такой подход позволяет существенно увеличить удобство использования инструмента верификации.

В данной статье мы рассматриваем одно из направлений развития такого подхода к верификации программ, позволяющего достичь полноты решающей процедуры для практически значимого класса целевых утверждений, а именно для формул, описывающих утверждения об адресной арифметике в языке Си. Вначале мы формулируем абстрактную аксиоматическую теорию адресной арифметики для языка Си с учетом как практического опыта исправления различных известных ошибок, связанных с адресной арифметикой, в индустриальном коде на языке Си, так и спецификации операций адресной арифметики, приведенной в последних версиях стандарта языка. Формулируемая теория, однако, не полностью формализует все аспекты корректного использования адресной арифметики в языке Си, но ограничивается достаточно значимым фрагментом тех свойств указателей, которые не связаны с состоянием адресуемой памяти. Таким образом, в частности, теория не включает операцию разыменования и полную формализацию понятия валидного указателя. Эти аспекты формализации модели памяти сильно зависят от конкретного используемого подхода к ее моделированию, такому как сепарационная логика или использование динамических фреймов и остаются за рамками данной работы. Однако, формализованная в данной статье модель совместима с различными методами моделирования памяти и полностью формализует понятия блока памяти (или происхождения (*provenance*) указателя), адреса, включая отсутствие переполнения, нулевого указателя, относительного выравнивания указателя на элемент массива и отчасти, понятие валидности, так как для невалидного указателя невозможно в общем случае показать отсутствие переполнения адреса.

1.1 Примеры известных ошибок при работе с указателями

Для формулирования теории ограниченной адресной арифметики рассмотрим вначале практические примеры известных ошибок, показывающие необходимость моделирования

блока памяти (происхождения) указателя, корректного соотнесения блока с адресом, а также проверки переполнения адреса в операциях адресной арифметики.

Вначале рассмотрим соотношение понятия указателя и адреса в языке Си, к формализации которого можно прийти, рассмотрев два важных примера работы оптимизирующих компиляторов, один из которых реально встречался в промышленном коде. Этот пример из исправления, включенного в одну из версий ядра ОС Linux [2]:

```
extern struct builtin_fw __start_builtin_fw[];
extern struct builtin_fw __end_builtin_fw[];
...
for (b_fw = __start_builtin_fw; b_fw != __end_builtin_fw; b_fw++) {
...
```

Здесь условие входа в цикл **for** оптимизировалось компилятором в тождественную истину, что впоследствии вызывало выход за границу выделенной страницы памяти. Такая семантика сравнения указателей в данном фрагменте совместима со стандартом языка Си, так как поведение компилятора при сравнении указателей на разные выделенные объекты в памяти (в данном случае массивы `__start_builtin_fw` и `__end_builtin_fw`), за исключением специального случая непосредственно следующих друг за другом объектов, является неопределенным поведением [3] и поддается произвольной трактовке в каждой конкретной реализации компилятора. Этот пример показывает, что понятие указателя в языке Си не сводится к понятию адреса, так как любой адрес в принципе может быть получен из любого другого с помощью операции адресной арифметики, а именно прибавления положительного или отрицательного смещения. Указатели же на элементы разных объектов в памяти с точки зрения компилятора рассматриваются как заведомо различные (не равные) независимо от прибавления к ним произвольного смещения. Это свойство указателей еще более наглядно демонстрирует следующий пример:

```
#include <stdio.h>
int main(void) {
    int a, b;
    int *p = &a;
    int *q = &b - 1;
    printf("%p %p %d", p, q, p == q);
    return 0;
}
```

В этом примере результатом работы такой программы в некоторых совместимых со стандартом ANSI C компиляторах со включенными оптимизациями (например, GCC 11.1.1 с опцией `-O3`) может быть, например, такая строка:

```
0x7ffe32e8ece8 0x7ffe32e8ece8 0
```

Видно, что хотя значения указателей `p` и `q`, представленные в виде адресов, совпадают, значение выражения `p == q` вычисляется компилятором в ложь (`0` в языке Си) вообще независимо от значений соответствующих адресов, потому что предполагается непересечение указателей на разные объекты памяти `a` и `b`, для которых не гарантируется никакое конкретное размещение (в том числе последовательное). Ориентируясь на приведенные примеры, можно предположить, что выделенный объект можно все же идентифицировать по его начальному адресу и представлять указатель в виде упорядоченной пары (a_0, a), где a — адрес, являющийся значением указателя, а a_0 — начальный адрес некоторого выделенного объекта, приписанного указателю компилятором. Однако с помощью небольшой модификации последнего примера можно получить еще один пример, опровергающий такую модель указателя:

```
#include <stdio.h>
int *f(int *u) {
```

```
    int b, *pb = &b;
    printf("%p %p %d", u, &b, u == &b);
    return pb;
}
int main(void) {
    int a, *b = f(&a);
    f(b);
    f(b);
    return 0;
}
```

При использовании компилятора Clang 12.0.1 с опцией `-O3` результатом работы такой программы могут быть, например, строки:

```
0x7ffd0382fa1c 0x7ffd0382fa18 0
0x7ffd0382fa18 0x7ffd0382fa18 0
0x7ffd0382fa18 0x7ffd0382fa18 0
```

Здесь, в отличие от предыдущих примеров, не используется прибавление к указателю смещения, а начальные адреса объектов в памяти совпадают. Но один из объектов, на который указывает указатель из параметра функции, уже не является выделенным. Таким образом, указатель на ранее выделенный объект считается всегда отличным от указателя на любой другой выделенный (в текущем состоянии или ранее) объект, независимо от начального адреса каждого из этих объектов. Выразить такое свойство можно, в частности, с помощью модели указателя в виде упорядоченной пары (l, a), где a — как и ранее — значение адреса, содержащееся в указателе, а l — уникальный идентификатор выделенного блока, который присваивается при любом выделении памяти и никогда не повторяется.

1.2 Требуемые понятия

Выделим теперь понятия, которые могут использоваться для спецификации операций над указателями, и определим, какие из них будут представлены в разрабатываемой модели адресной арифметики и каким образом. Помимо хранимого в указателе адреса и идентификатора блока памяти, который не меняется при сдвиге указателя на любое смещение, при описании операций над указателями, как в стандарте ANSI C, так и в повседневной программистской практике, часто используются следующие понятия:

- *Размер* выделенного объекта (в смысле количества элементов). Это понятие часто используется неявно в виде упоминания «последнего элемента массива». Проблема формализации размера выделенного объекта в отрыве от соответствующей модели памяти в том, что этот размер зависит от рассматриваемого состояния программы и меняется при выделении или освобождении памяти, занимаемой объектом. Поэтому в теорию адресной арифметики, формализуемую в данной статье, это понятие не может быть включено явно. Однако, как будет ясно далее при рассмотрении семантики операций адресной арифметики, проверки переполнения адреса потребуют дополнительных предусловий вида $0 \le a$ и $a \le A$, где a — адрес, хранящийся в указателе. Эти предусловия могут выводиться, в частности, из условия выделенности адресуемой указателем памяти, которое формализуется уже в рамках выбранной модели памяти, а не адресной арифметики, и связано с текущим размером выделенного объекта.
- Начальный (*базовый*) адрес объекта. Это понятие возникает, например, при формализации спецификаций функций управления памятью (таких как `malloc` и `free`) и, в отличие от размера объекта, не меняется в ходе выполнения программы и, таким образом, не зависит от ее состояния. Так как в силу семантики вложенных объектов, принятой в языке Си и предполагающей непосредственное плоское размещение вложенных объектов одного за другим, в полной мере различными объектами в

программе могут считаться только объекты, изначально выделенные в разных точках программы (в разных объявлениях переменных либо при различных вызовах функций управления памятью), базовый адрес объекта фактически становится атрибутом соответствующего ему блока памяти. Поэтому в теории адресной арифметики базовый адрес формализуется как функция от идентификатора выделенного блока. Будем далее обозначать эту функцию от блока l как $\boldsymbol{base}\,(l)$.
- *Смещение* объекта относительно базового адреса. В ситуации, когда основные операции над указателями (сдвиг, вычитание, сравнение) определены только для указателей на элементы одного изначально выделяемого базового массива объектов и указателей на вложенные в элементы этого массива подобъекты, чаще всего имеет смысл говорить не об абсолютном адресе объекта в памяти, а о его смещении от начала (базового адреса) выделенного объекта (массива). При таком подходе адрес объекта может быть всегда представлен как сумма $\boldsymbol{address}(p) = \boldsymbol{base}(\boldsymbol{block}(p)) + \boldsymbol{offset}(p)$, где p — указатель, $\boldsymbol{block}(p)$ – идентификатор блока памяти, приписанного указателю p, $\boldsymbol{offset}(p)$ – смещение указателя p от базового адреса (целое число байт), а $\boldsymbol{address}(p)$ – значение адреса, хранящееся в указателе p.
- Еще одно понятие неявно возникает при формализации такого термина, как «элементы одного массива». Чтобы понять существенное отличие этого понятия от элементов одного блока памяти, рассмотрим для примера следующий массив:

```
struct s {
   int b[2];
   char c;
   int c[2];
} a[2];
```

В этом примере можно видеть по крайней мере четыре различных разновидности указателей типа `int *`: `&a[0].b[0]`, `&a[0].b[1]`, `&a[0].c[0]` и `&a[1].b[0]`. При этом про пары указателей (`&a[0].b[0]`, `&a[0].b[1]`) и (`&a[0].b[0]`, `&a[1].b[0]`) имеет смысл говорить как об указателях на элементы одного массива (внутреннего массива `b`, либо внешнего массива `a`, а про оставшиеся пары (`&a[0].b[0]`, `&a[0].c[0]`), (`&a[0].b[1]`, `&a[0].c[0]`), (`&a[0].b[1]`, `&a[1].b[0]`) и (`&a[0].c[0]`, `&a[1].b[0]`) — в общем случае нет. Для формализации такого понимания элементов одного массива хорошо подходит понятие *выравнивание* указателя, определенное как остаток от деления его абсолютного адреса, либо смещения относительно базового адреса, на размер типа указателя (в данном случае `int`). Если считать, что смещение поля `c` структуры `s` относительно ее поля `b` фиксировано некоторой неизвестной константой d, а размер структуры `s` равен также неизвестной фиксированной константе s, то выравнивания указателей в соответствующих парах будут выражаться как $(0, 0)$, $(s \,\%\, sizeof\,(int), s \,\%\, sizeof\,(int))$ соответственно для первых двух пар и как $(0, d \,\%\, sizeof\,(int))$, $(0, d \,\%\, sizeof\,(int))$, $(0, (s + d) \,\%\, sizeof\,(int))$ и $(d \,\%\, sizeof\,(int), (s + d) \,\%\, sizeof\,(int))$ соответственно для четырех остальных. В общем случае константы s и d могут быть подобраны таким образом, что выравнивания для последних четырех (неупорядоченных) пар указателей не совпадут. Таким образом, остаток от деления смещения на размер указателя дает желаемую семантику.

Перед формализацией операций над указателями во введенной модели указателей как упорядоченных пар (l, a) необходимо отдельно выделить понятие нулевого указателя. Это представляет собой некоторую трудность, потому что с одной стороны такой указатель должен быть отличен от любого указателя на выделенный объект, с другой, поскольку в реализации указатели представляются только хранимыми в них адресами, в принципе нулевой указатель может быть получен из указателей на различные выделенные объекты с помощью сдвига на соответствующее отрицательное смещение, а помимо этого любые два

нулевых указателя должны считаться равными друг другу. Чтобы обойти трудность формализации нулевого указателя, применим верхнее приближение, оставив поведение операции сдвига неопределенным при получении указателя с нулевым адресом, а также исключив из определений всех операций, кроме сравнения на равенство, операции над любыми указателями, имеющими нулевой адрес. Такая формализация не полностью покрывает все возможные поведения, разрешенные стандартом, в частности, она предполагает, что нулевой указатель хранит нулевой адрес, но на практике это соответствует поведению практически всех существующих реализаций языка. Кроме этого, такая формализация не исключает семантику, при которой помимо указателей на объекты (валидные и невалидные) и нулевого указателя могут существовать и другие указатели с нулевым адресом, которые тем не менее отличны от нулевого указателя.

1.3 Семантика основных операций

Определим теперь семантику основных операций с указателями с использованием введенных понятий и соответствующих им формальных обозначений:

1.3.1 Сдвиг указателя

Стандарт ANSI C (пункт 6.5.6(8)) определяет сложение указателей в предположении отсутствия переполнения адреса, полученного в результате сдвига. Это практически важное предположение, которое приводило к возникновению потенциально критических ошибок в промышленном коде [4]. Помимо отсутствия переполнения, операция сдвига указателя должна быть не определена для аргумента либо результата с нулевым адресом. Сдвиг указателя также порождает указатель на тот же объект памяти (без учета вложенности, то есть в нашей модели — блок), что и исходный указатель. Эти соображения приводят к следующей аксиоматической формализации операции сдвига указателя ($+_p$):

$$\forall p\ i.\ \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{range}(p, i) \longrightarrow p \,\triangle\, p +_p i,$$
$$\forall p\ i.\ \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{range}(p, i) \longrightarrow \boldsymbol{offset}\,(p +_p i) = \boldsymbol{offset}\,(p) + s \times i, \text{где}$$
$$\boldsymbol{range}(p, i) \equiv 0 < \boldsymbol{address}(p) + s \times i \wedge \boldsymbol{address}(p) + s \times i \le A,$$
$$p \,\triangle\, q \equiv \boldsymbol{block}(p) = \boldsymbol{block}(q).$$

Несмотря на то, что такая формализация не полностью ограничивает операцию сдвига указателя в соответствии со стандартом, не требуя выделенности соответствующего объекта в памяти и не исключая выход за границу выделенного массива объектов более, чем на один элемент, на практике доказательство отсутствия переполнения (условие $\boldsymbol{range}(p, i)$) потребует использования условия выделенности аргумента или результирующего указателя, которое будет формализовано с использованием соответствующей модели памяти. Далее в статье будем рассматривать один тип указателей на некоторый фиксированный тип с ненулевым размером s и адресное пространство с максимально допустимым адресом A. Арифметика с указателями на типы других размеров может быть формализована аналогично, а приведение типа указателя может быть формализовано через комбинацию операций получения адреса ($\boldsymbol{address}(p)$), идентификатора блока $\boldsymbol{block}(p)$ (для исходного типа указателя) и операцию получения нового указателя $(l, a)_p$ (для целевого типа указателя), рассмотренную далее.

1.3.2 Разность указателей

Разность указателей определена в стандарте языка только для элементов одного массива. Учитывая приведенные соображения по поводу использования выравниваний для идентификации элементов одного массива и исключения из семантики операций нулевых

указателей, получим следующую аксиоматическую формализацию операции разности указателей ($-_p$):

$$\forall p\ q.\ p \,\triangle_0\, q \wedge p \parallel q \longrightarrow s \times (p -_p q) = \boldsymbol{offset}(p) - \boldsymbol{offset}(q), \text{где}$$
$$p \,\triangle_0\, q \equiv p \,\triangle\, q \wedge \boldsymbol{address}(p) \neq 0 \wedge \boldsymbol{address}(q) \neq 0,$$
$$p \parallel q \equiv \boldsymbol{align}(p) = \boldsymbol{align}(q).$$

Для формализации выравнивания ($\boldsymbol{align}(p)$) сразу учтем выразительные возможности целевой логики — комбинации теорий бескванторной линейной целочисленной арифметики и конгруэнтности с неинтерпретируемыми функциями (логика QF_UFLIA). В этой теории операцию получения остатка от деления на константу s можно выразить с помощью следующих аксиом (вопрос о полноте инстанцирования кванторов рассмотрен далее и является основным предметом рассмотрения данной статьи):

$$\forall p.\ \boldsymbol{offset}(p) = s \times \boldsymbol{quot}(p) + \boldsymbol{align}(p),$$
$$\forall p.\ 0 \le \boldsymbol{align}(p) \wedge \boldsymbol{align}(p) < s.$$

Здесь $\boldsymbol{quot}(p)$ – неинтерпретируемая функция, использованная для сколемизации квантора существования частного от деления.

1.3.3 Сравнение указателей на неравенство

В стандарте ANSI C сравнение указателей определено только для указателей на один и тот же выделенный объект памяти без учета вложенности, что соответствует условию $p \,\triangle\, q$. С учетом исключения нулевого указателя, для которого сравнение на неравенство не определено, получаем условие $p \,\triangle_0\, q$. Без ограничения общности будем рассматривать только одну операцию сравнения указателей на неравенство ($\le_p$):

$$\forall p\ q.\ p \,\triangle_0\, q \longrightarrow p \le_p q = \boldsymbol{offset}(p) \le \boldsymbol{offset}(q).$$

1.3.4 Сравнение указателей на равенство

Так как в стандарте разрешается сравнение указателей на два различных выделенных объекта памяти, а также сравнение на равенство с нулевым указателем (в том числе нулевых указателей друг с другом), для операции сравнения на равенство не удается формализовать никаких значимых предусловий. Условие выделенности обоих адресуемых объектов не может быть формализовано в отрыве от используемой модели памяти. Поэтому для сравнения указателей на равенство предлагается непосредственно использовать равенство соответствующих значений из теории конгруэнтности (=). Для операций сравнения указателей на равенство, используемых в коде программы, можно отдельно задать предусловие выделенности для случая сравнения указателей на различные объекты (с использованием модели памяти). В то же время формализованная теория адресной арифметики (даже без этого предусловия) дает достаточно точное приближение возможных поведений программ после компиляции, учитывая приписываемые компилятором указателям идентификаторы выделенных объектов в памяти. Кроме этого, точное определение (не)выделенности адресуемого объекта достаточно сложно в реализации (точнее, в общем случае алгоритмически неразрешимо), чтобы вероятность намеренного использования оптимизирующим компилятором неопределенного поведения, разрешенного стандартом для сравнения с указателем на невыделенный объект, была очень невысокой.

1.3.5 Получение нового указателя: по адресу и идентификатору блока

Эта операция необходима для полноты определения теории адресной арифметики, а также для выражения операции приведения типа указателя и операции приведения к указателю значения целочисленного типа. Несмотря на то, что в исходном коде программы такая операция фактически используется для одного аргумента — адреса, используемая в языке семантика указателей фактически требует аннотирования операций приведения типа блоком

памяти полученного указателя. В рассматриваемой формализации мы не требуем никакого конкретного способа определения результирующего блока памяти, оставляя возможность для различных конкретных реализаций в инструментах верификации. Формализация операции получения нового указателя $((l,a)_p)$ должна устанавливать два основных свойства полученного указателя — его адрес и блок памяти, а также устанавливать полноту этих свойств, полностью определяющих значение указателя:

$$\forall l\ a.\ 0 \le a \wedge a \le A \rightarrow \mathbf{block}((l,a)_p) = l,$$
$$\forall l\ a.\ 0 \le a \wedge a \le A \rightarrow \mathbf{address}((l,a)_p) = a,$$
$$\forall p.\ \left(\mathbf{block}(p), \mathbf{address}(p)\right)_p = p.$$

Для полноты аксиоматического определения теории адресной арифметики остается задать последнюю аксиому, отражающую ограниченность размера адресного пространства:

$$\forall p.\ 0 \le \mathbf{address}(p) \wedge \mathbf{address}(p) \le A.$$

Таким образом, в данной статье рассматривается аксиоматическая формулировка семантики теории ограниченной (то есть с ограниченным размером адресного пространства) адресной арифметики, включающей в себя 8 основных операций: $(l,a)_p$, $\mathbf{block}(p)$, $\mathbf{offset}(p)$, $\mathbf{align}(p)$, $\mathbf{base}(p)$, $p +_p i$, $p -_p q$ и $p \le_p q$. В статье рассматривается комбинация этой теории с логикой QF_UFLIA, определенной в формате SMT-LIB. Соответствующая задача о выполнимости формул в полученной комбинации теорий решается с помощью трансляции исходных формул в соответствующие равновыполнимые формулы в базовой логике QF_UFLIA, которая поддерживается как большинством современных SMT-решателей, так и процедурой воспроизведения доказательств, реализованной в системе Isabelle/HOL. Оставшаяся часть статьи практически целиком посвящена формулировке и доказательству корректности и полноты соответствующей процедуры трансляции формул из полученной теории ограниченной адресной арифметики (далее — BPA) в логику QF_UFLIA.

Сформулированная решающая процедура для теории BPA также была реализована в виде метода в системе автоматизированных доказательств Isabelle/HOL. Этот метод состоит в инстанцировании аксиом теории BPA термами из исходной формулы в соответствии с представленной в статье полной процедурой трансляции, интерпретации полученной формулы в логике QF_UFLIA (при этом символы теории BPA становятся неинтерпретируемыми), применении соответствующей существующей полной решающей процедуры с помощью вызова SMT-решателя и последующего восстановления доказательства средствами системы Isabelle/HOL. В свою очередь, доказательство полноты предложенного метода основывается на преобразовании полученной от SMT-решателя модели в логике QF_UFLIA в модель теории BPA.

Как отмечалось ранее в предыдущей статье [5], нам не требуется вносить изменения в инструменты воспроизведения доказательств системы Isabelle/HOL и SMT-решатель. Кроме этого, инстанцирование аксиом исходной теории в соответствии с предложенной процедурой увеличивает исходный размер анализируемой формулы не более, чем линейно. Основной целью нашей работы являлось доказательство полноты и корректности предложенной процедуры преобразования формул из теории BPA в логику QF_UFLIA.

2. Существующие модели адресной арифметики

Как правило, при моделировании семантики практически значимых фрагментов языка Си модель адресной арифметики рассматривается совместно с моделью памяти, то есть с операциями чтения и записи значения по указателю. Тем не менее, в тех моделях семантики языка (в частности, моделях памяти), которые учитывают возможность использования разрешенных стандартом оптимизаций, как правило, легко может быть выделен фрагмент, соответствующий формализованной в данной статье теории BPA. Среди четырех

рассмотренных нами работ [6-9] по моделированию памяти для языка Си фрагмент, соответствующий теории BPA, может быть выделен в двух работах — [7] и [9]. В обеих этих моделях для представления указателей используются пары вида (l,o), отличные от представленных в этой статье указателей вида (l,a) только непосредственным использованием смещения вместо адреса. В таких моделях можно определить соответствующие понятия смещения и базового адреса так, что аксиомы теории BPA будут выполнены (доказанны) как леммы (теоремы) соответствующей модели памяти. В таком случае теория BPA будет являться фрагментом модели памяти, и ее семантика будет надежно приближать семантику всех присутствующих в ней операций адресной арифметики, кроме сравнения указателей на равенство. Предусловие операции сравнения указателей на равенство не выражается в понятиях, формализованных в рамках самой теории BPA (в ней нет понятия валидности указателя), и поэтому приближается в ней снизу, то есть определяется семантически более сильная операция. Это не мешает, однако, формализовать также и корректное (верхнее) приближение операции сравнения указателей на равенство (с использованием формализованного в модели памяти понятия валидности), но для полученной в результате теории не будет в полной мере работать процедура, представленная в данной статье. Это означает, что в рамках моделей памяти, представленных в работах [7] и [9] представленная в этой статье решающая процедура для теории BPA будет неполной только для случаев сравнения на равенство указателей на два различных блока памяти, по крайней мере один из которых не является валидным. Тем не менее, так как такие случаи на практике достаточно редки, полная решающая процедура для теории BPA может быть существенно использована как для упрощения доказательств в подобных моделях памяти, так и для поиска контрпримеров утверждений в рамках этой теории, для которых сохраняется полнота.

Представленные в двух других работах ([6] и [8]) модели памяти делают предположения о семантике адресной арифметики, существенно отличающиеся как от семантики теории BPA, так и от семантики самого языка Си с точки зрения оптимизирующего компилятора. В работе [6] (использованной на практике при верификации кода микроядра L4[11]) не рассматриваются уникальные идентификаторы блоков, приписываемые различным объектам в памяти программы оптимизирующим компилятором. Указатели непосредственно соответствуют хранимым адресам. В работе [8] значения адресов объектов в памяти программы не ограничены сверху и могут быть представлены математическими целыми (предполагается отсутствие переполнений значения указателя). Таким образом, обе этих модели упускают существенные на практике аспекты семантики указателей в языке Си. Другие примеры неочевидного поведения программ при использовании операций вычитания и присваивания указателей, соответствующие тем не менее стандарту языка C11, приведены в работе [10].

3. Основные определения

В стандарте SMT-LIB [13], QF_UFLIA — это логика бескванторных формул с неинтерпретируемыми символами и равенством в комбинации теорий линейной целочисленной арифметики и неинтерпретируемых функций. Логику QF_UFLIA можно рассматривать как комбинацию логик QF_LIA и QF_UF. QF_LIA обозначают замкнутые бескванторные формулы с равенством в теории линейной целочисленной арифметики (LIA). В сигнатуру этой теории входят следующие функциональные символы: $\{+, c \times, \le\}$, где c — целочисленная константа. QF_UF обозначает замкнутые бескванторные формулы с равенством в теории неинтерпретируемых функций.

Теорию ограниченной адресной арифметики BPA будем задавать аксиоматически как расширение логики QF_UFLIA. В сигнатуру данной теории включим следующие функциональные символы:

$\{\cdot \le_p \cdot,\ \cdot +_p \cdot,\ \cdot -_p \cdot,\ \mathbf{block}(\cdot),\ \mathbf{base}(\cdot),\ \mathbf{offset}(\cdot),\ \mathbf{align}(\cdot),\ (\cdot,\cdot)_p\}$.

Термы в этой теории могут быть двух различных сортов – указатели и целые. Семантика предполагает моделирование указателей как упорядоченных пар — (целое число, целое число), где первый элемент пары соответствует уникальному идентификатору блока памяти, а второй — хранимому в указателе адресу. Произвольный указатель можно таким образом получить при помощи соответствующей функции $(\cdot,\cdot)_p$. Неформальная семантика остальных функциональных символов была объяснена ранее во введении.

Для решения задачи о выполнимости формул в теории BPA будем использовать процедуру трансляции в логику QF_UFLIA. Процедура преобразования исходной формулы F состоит из последовательного инстанцирования аксиом теории BPA. Инстанцированием аксиомы A для некоторой формулы F будем называть взятие конъюнкции F с экземпляром аксиомы A, в котором вместо всех подкванторных переменных подставлены некоторые подтермы формулы F соответствующего сорта. Подтермы формулы F выбираются согласно правилам процедуры инстанцирования — триггерам, задаваемым отдельно для каждой аксиомы теории. Инстанцирование выполняется последовательно для всех аксиом и подтермов исходной формулы F, которые удовлетворяют заданным триггерам. В результате получается формула F^*, которая интерпретируется в логике QF_UFLIA. На формуле F^* запускается решающая процедура SMT-решателя для этой логики, которая является полной. В случае невыполнимости формулы F^* в результате получается доказательство, которое затем воспроизводится (сертифицируется) средствами системы Isabelle/HOL [14], [15]. В случае выполнимой формулы полнота решающей процедуры гарантирует существование модели формулы F^*.

Для доказательства полноты процедуры инстанцирования рассматриваем случай, когда мы получили некоторую модель R формулы F^*. Назовем модель R формулы F^* реализацией формулы F. В данной статье мы показываем, как из реализации R может быть восстановлена полная модель M исходной формулы F в теории BPA.

Дадим формальное определение теории BPA. Пусть $\mathbb{Z}$ – множество целых чисел, $\mathbb{P}$ — множество указателей, $\Sigma = \{+_p, \times_p, \le_p, \mathbf{block}, \mathbf{base}, \mathbf{offset}, \mathbf{align}, (\cdot,\cdot)_p\}$ — сигнатура теории BPA. Аксиомы теории BPA могут быть записаны в логике QF_UFLIA. Они представлены на рис. 1.

4. Процедура инстанцирования

Зададим процедуру трансляции формулы F в теории BPA в равновыполнимую формулу F^* в логике QF_UFLIA. Данная процедура использует только аксиомы теории *BPA*, представленные на рис. 1, и определяет для их инстанцирования триггеры следующим образом:

- (A_p) инстанцируется всеми указательными термами p в формуле F;
- $(A_{\%})$ инстанцируется всеми указательными термами p в формуле F;
- $(A_{\in})$ инстанцируется всеми указательными термами p в формуле F;
- (A_o) инстанцируется всеми указательными термами p в формуле F;
- $(A_{\le})$ инстанцируется термами p и q для любого терма вида $p \le_p q$ в формуле F;
- (A_{-}) инстанцируется термами p и q для любого терма вида $p -_p q$ в формуле F;
- (A_{Δ}) инстанцируется термами p и i для любого терма вида $p +_p i$ в формуле F;
- (A_{+}) инстанцируется термами p и i для любого терма вида $p +_p i$ в формуле F;
- (A_a) инстанцируется термами l и a для любого терма вида $(l,a)_p$ в формуле F;
- (A_b) инстанцируется термами l и a для любого терма вида $(l,a)_p$ в формуле F.

Приведенные правила называются триггерами соответствующих аксиом.

$$\Sigma = \{+_p, -_p, \le_p, \mathbf{block}, \mathbf{base}, \mathbf{offset}, \mathbf{align}, (\cdot,\cdot)_p\},$$
$$p, q \in \mathbb{P};\ l, a, i, j \in \mathbb{Z}.$$
$$\mathbf{block}(p) \in \mathbb{Z}, \mathbf{base}(p) \in \mathbb{Z}, \mathbf{offset}(p) \in \mathbb{Z}, \mathbf{align}(p) \in \mathbb{Z}, (l,a)_p \in \mathbb{P}.$$

Введем обозначения:

$$\mathbf{address}(p) = \mathbf{base}(\mathbf{block}(p)) + \mathbf{offset}(p),$$
$$p \parallel q \equiv \mathbf{align}(p) = \mathbf{align}(q),$$
$$p\ \Delta\ q \equiv \mathbf{block}(p) = \mathbf{block}(q),$$
$$p\ \Delta_0\ q \equiv p\ \Delta\ q \wedge \mathbf{address}(p) \neq 0 \wedge \mathbf{address}(q) \neq 0$$
$$\mathbf{range}(p,i) \equiv$$
$$0 < \mathbf{address}(p) + s \times i \wedge \mathbf{address}(p) + s \times i \le A, \text{ где } s, A \in \mathbb{Z} - \text{константы}.$$

Далее определим аксиомы:

$$\forall p,q \in \mathbb{P}.\ p\ \Delta_0\ q \Rightarrow (p \le_p q) \leftrightarrow \mathbf{offset}(p) \le \mathbf{offset}(q), \quad (A_{\le})$$
$$\forall p \in \mathbb{P}.\ \left(\mathbf{block}(p), \mathbf{address}(p)\right)_p = p, \quad (A_p)$$
$$\forall p \in \mathbb{P}.\ 0 \le \mathbf{align}(p) \wedge \mathbf{align}(p) \le s - 1, \quad (A_{\%})$$
$$\forall p \in \mathbb{P}.\ 0 \le \mathbf{address}(p) \wedge \mathbf{address}(p) \le A, \quad (A_{\in})$$
$$\forall p,q \in \mathbb{P}.\ p\ \Delta_0\ q \wedge p \parallel q \Rightarrow$$
$$s \times (p -_p q) = \mathbf{offset}(p) - \mathbf{offset}(q), \quad (A_{-})$$
$$\forall p \in \mathbb{P}.\ \mathbf{offset}(p) = s \times \mathbf{quot}(p) + \mathbf{align}(p), \quad (A_o)$$
$$\forall i \in \mathbb{Z}.\ \forall p \in \mathbb{P}.\ \mathbf{address}(p) \neq 0 \wedge \mathbf{range}(p,i) \Rightarrow p +_p i\ \Delta\ p \quad (A_{\Delta})$$
$$\forall i \in \mathbb{Z}.\ \forall p \in \mathbb{P}.\ \mathbf{address}(p) \neq 0 \wedge \mathbf{range}(p,i) \Rightarrow$$
$$\mathbf{offset}(p +_p i) = \mathbf{offset}(p) + s \times i \quad (A_{+})$$
$$\forall l, a \in \mathbb{Z}.\ 0 \le a \wedge a \le A \Rightarrow \mathbf{address}\left((l,a)_p\right) = a \quad (A_a)$$
$$\forall l, a \in \mathbb{Z}.\ 0 \le a \wedge a \le A \Rightarrow \mathbf{block}\left((l,a)_p\right) = l \quad (A_b)$$

Рис. 1. Аксиоматика теории ограниченной адресной арифметики
Fig. 1. Axioms defining the theory of bounded pointer arithmetic

Обозначим за F^* формулу F после выполнения алгоритма инстанцирования аксиом. Обозначим за $F^{\le}$ множество экземпляров аксиом, полученных подстановкой термов вместо подкванторных переменных p и q в аксиоме $(A_{\le})$ для любого терма вида $p \le_p q$ в формуле F. Аналогично определим множества F^p для (A_p), $F^{\%}$ для $(A_{\%})$, $F^{\in}$ для $(A_{\in})$, F^o для (A_o), F^{Δ} для (A_{Δ}), F^+ для (A_+), F^- для (A_-) и, наконец, F^a для (A_a) и F^b для (A_b). Полученная после инстанцирования формула F^* тогда может быть записана следующим образом:

$$F^* = F \wedge \wedge F^p \wedge \wedge F^{\%} \wedge \wedge F^{\in} \wedge \wedge F^o \wedge \wedge F^{\Delta} \wedge \wedge F^+ \wedge \wedge F^- \wedge \wedge F^{\le} \wedge \wedge F^a \wedge \wedge F^b.$$

Формула F^* интерпретируется в логике QF_UFLIA, так что все символы теории BPA в ней не интерпретируются. Так как формула F^* получается из F только с помощью инстанцирования аксиом теории BPA, из выполнимости формулы F в BPA следует выполнимость F^* в логике QF_UFLIA, что соответствует корректности представленной процедуры трансляции.

5. Доказательство полноты

Рассмотрим произвольную формулу F в теории BPA, её трансляцию F^*, полученную через процедуру инстанцирования аксиом, и модель R, полученную для трансляции F^* в логике QF_UFLIA, которую мы называем реализацией. Указательные термы далее будем обозначать

буквами p и q, а целочисленные термы — буквами l, a, i и j. Доказательство полноты осуществляется с помощью восстановления модели M исходной формулы F в теории BPA из реализации R.

Для начала приведем несколько вспомогательных лемм.

Лемма 1. *Терм вида* $\mathbf{block}(p)$ *принадлежит формуле* F^* *тогда и только тогда, когда соответствующий указательный терм* p *принадлежит* F.

Лемма 2. *Терм вида* $\mathbf{address}(p)$ *принадлежит формуле* F^* *тогда и только тогда, когда соответствующий адресный терм* p *принадлежит* F.

Лемма 3. *Терм вида* $(l, a)_p$ *принадлежит формуле* F^* *тогда и только тогда, когда либо он исходно принадлежит* F, *либо подтерм* l *имеет вид* $\mathbf{block}(p)$, *а подтерм* a — *вид* $\mathbf{address}(p)$, *где соответствующий указательный терм* p *принадлежит* F.

Доказательство этих лемм осуществляется непосредственным перебором приведенных правил инстанцирования.

Далее дадим несколько дополнительных обозначений, которые мы будем использовать в доказательствах. Определим образ подмножества X множества A при отображении с помощью функции $f: A \to B$ как $f[X]$. Введем следующее обозначение:

$\{f(x) \mid P(x)\} \equiv f[\{x \mid P(x)\}]$, где $\{x \mid P(x)\}$ — множество термов x, удовлетворяющих предикату $P(x)$.

Далее определим следующие множества:

$P^R \equiv \{p^R \mid \mathbf{address}(t) \in F^*\}$,

$L^R \equiv \{(l^R,\ a^R) \mid (b, a)_p \in F^*\}$,

$L^A \equiv (\mathbb{Z} \times [0, A]) \backslash L^R$, где, p^R, l^R и a^R обозначают означивания соответствующих термов p, l и a в реализации R. Также введем функцию $(\cdot)_\perp$ преобразования указателя в пару, которая является следующим сокращением: $p_\perp \equiv (\mathbf{block}(p), \mathbf{address}(p))$.

Реализация R уже содержит частичные модели функций $(\cdot,\cdot)_p$ и $\cdot_\perp$ на соответствующих множествах L^R и P^R. Рассмотрим функции $(\cdot,\cdot)_p^R$ и $\cdot_\perp^R$, которые будем считать реализациями соответствующих функций $(\cdot,\cdot)_p$ и $\cdot_\perp$ на множествах L^R и P^R соответственно. Докажем для них следующие свойства:

Лемма 4. $[P^R]_\perp^R \subseteq L^R \cap (\mathbb{Z} \times [0, A])$. Показывается с помощью лемм 1–3.

Лемма 5. $[L^R]_p^R \subseteq P^R$. Показывается с помощью лемм 1 и 2.

Лемма 6. $\cdot_\perp^R$ *инъективна на множестве* P^R. Следует из лемм 1–5 и правила инстанцирования аксиомы (A_p).

Лемма 7. $\cdot_\perp^R$ *сюръективно отображает* P^R *на множество значений* $L^R \cap (\mathbb{Z} \times [0, A])$. Следует из лемм 1–5 и правила инстанцирования аксиомы $(A_\in)$.

Лемма 8. $\cdot_\perp^R$ *является биекцией между множествами* P^R *и* $L^R \cap (\mathbb{Z} \times [0, A])$, *а функция* $(\cdot,\cdot)_p^R$ *на множестве* $L^R \cap (\mathbb{Z} \times [0, A])$ *является обратной к ней.*

Первая часть утверждения о том, что $\cdot_\perp^R$ является биекцией непосредственно следует из лемм 6 и 7, а обратная функция может быть однозначна охарактеризована уравнением $(\cdot_\perp^R)^{-1}(p')_\perp^R = p'$ для любого $p' \in P^R$. Но для каждого такого p' верно, что $p' = p^R$, где $\mathbf{address}(p) \in F^*$. Из леммы 2 следует, что $p \in F$. Далее, согласно правилу инстанцирования аксиомы A_p верно, что $(\mathbf{block}(p), \mathbf{address}(p))_p = p$. Таким образом, обратная к биективной $\cdot_\perp^R$ функция $(\cdot_\perp^R)^{-1}$ совпадает с $(\cdot,\cdot)_p^R$.

Получим ситуацию, изображенную на рис. 2, где $\cdot_\perp^R$ является биекцией между P^R и $L^R \cap (\mathbb{Z} \times [0, A])$ (на рис. 2 это множество обозначено двойной штриховкой), а $(\cdot,\cdot)^R$ определена на множестве L^R и является обратной к $\cdot_\perp^R$ на P^R. Рассмотрим теперь множество $L^A \equiv (\mathbb{Z} \times [0, A]) \backslash L^R$. Оно имеет не более, чем счетную мощность. Выберем произвольно

соответствующее (конечное, либо счетное) число различных элементов из какого-либо счетного домена и обозначим полученное таким образом множество элементов через $\mathbb{Z}'_p$. Теперь в качестве домена указателей в восстановленной модели M возьмем множество $P^R \cup \mathbb{Z}'_p$. Обозначим его $\mathbb{P}^M$. Так как множества L^A и $\mathbb{Z}'_p$ по построению имеют одинаковую мощность, между ними существует биекция. Произвольно выберем такую биекцию, определенную на множестве $\mathbb{Z}'_p$ и назовем ее $\cdot'_\perp$, а обратную к ней функцию, определенную на множестве L^A — $(\cdot,\cdot)'_p$. Введем сокращения $p'_\perp = (\mathbf{block}'(p), \mathbf{address}'(p))$ и определим

$$\mathbf{offset}'(p) = \begin{cases} \mathbf{address}'(p) - \mathbf{base}^R(\mathbf{block}'(p)), & \mathbf{block}'(p) \in \{l^R \mid \mathbf{base}(l) \in F^*\} \\ \mathbf{address}'(p), & \text{иначе} \end{cases}$$

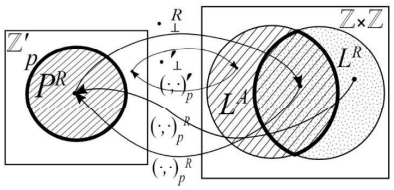

Рис. 2. Расширение биекции между множествами P^R *и* $L^R \cap (\mathbb{Z} \times [0, A])$ *на множества* $\mathbb{P}^M = P^R \cup \mathbb{Z}'_p$ *и* $\mathbb{Z} \times [0, A]$

Fig. 2. Extending the bijection between P^R *and* $L^R \cap (\mathbb{Z} \times [0, A])$ *to the whole sets* $\mathbb{P}^M = P^R \cup \mathbb{Z}'_p$ *and* $\mathbb{Z} \times [0, A]$

Теперь мы готовы ввести определение восстановленной из реализации R полной модели M исходной формулы F в теории BPA, которое представлено на рис. 3. Далее докажем необходимые свойства этого определения.

Лемма 11. *Восстановленная модель* M *на рисунке 3 однозначно определена.*

Восстановленная модель M, показанная на рис. 3, включает в себя определение домена $\mathbb{P}^M$ всех указателей, а также определения моделей для всех функций из сигнатуры теории BPA. Эти функции могут возвращать элементы двух сортов — целые числа и указатели.

Таким образом, мы должны показать, что функции, возвращающие указатели, определенные на рисунке, действительно отображают любую комбинацию своих аргументов, взятых из соответствующих доменов, в элементы домена указателей $\mathbb{P}^M$.

Рассмотрим случаи для функции $(\cdot,\cdot)_p^M$. В первом случае $(l, a)_p^R \in P^R$ для любой пары $(l, a) \in L^R$ согласно лемме 7. Во втором случае $(l, a)'_p \in \mathbb{Z}_p{}' \subseteq \mathbb{P}^M$ по построению домена $\mathbb{P}^M$. В третьем случае $\epsilon\mathbb{P}^M \subseteq \mathbb{P}^M$ по определению эпсилон-оператора Гильберта ϵ (выбор произвольного элемента из непустого множества), так как в общем случае либо $\mathbb{Z}_p{}'$, либо P^R непусты. Оставшаяся указательная функция $+_p^M$ в модели M однозначно определена с использованием функции $(\cdot,\cdot)_p^M$.

Лемма 12. *Аксиомы* (A_p), $(A_\in)$, $(A_{\mathbb{N}})$ *и* (A_O) *теории BPA выполнены в модели* M.

Лемма 12. *Аксиомы* $(A_\leq)$ *и* (A_-) *теории BPA сохраняются в модели* M.

Лемма 14. *Аксиомы* (A_Δ) *и* (A_+) *теории BPA сохраняются в модели* M.

Лемма 15. *Аксиомы* (A_b) *и* (A_a) *теории BPA сохраняются в модели* M.

Лемма 16. *Модель* M *может быть расширена неинтерпретируемыми константами, которые содержатся в* F *так, что для любого подтерма* $t \in F$ *его интерпретации в модели* M *и в реализации* R *совпадают, то есть* $t^M = t^R$.

$$\mathbb{P}^M = \mathrm{P}^R \cup \mathbb{Z}'_p,$$

$$(l, a)_p^M = \begin{cases} (l, a)_p^R, & (l, a) \in L^R \\ (l, a)'_p, & (l, a) \notin L^R, a \in [0, A] \\ \epsilon\mathbb{P}^M, & (l, a) \notin L^R, a \notin [0, A] \end{cases}$$

$$\mathbf{block}^M(p) = \begin{cases} \mathbf{block}^R(p), & p \in P^R \\ \mathbf{block}'(p), & p \notin P^R \end{cases}$$

$$\mathbf{offset}^M(p) = \begin{cases} \mathbf{offset}^R(p), & p \in P^R \\ \mathbf{offset}'(p), & p \notin P^R \end{cases}$$

$$\mathbf{base}^M(l) = \begin{cases} \mathbf{base}^R(l), & l \in \{l^R \mid base(l) \in F^*\} \\ 0, & l \notin \{l^R \mid base(l) \in F^*\} \end{cases}$$

$$\mathbf{align}^R(p) = \begin{cases} \mathbf{align}^R(p), & p \in P^R \\ \mathbf{offset}'(p)\ \%\ s, & p \notin P^R \end{cases}$$

$$\mathbf{quot}^M(p) = \begin{cases} \mathbf{quot}^R(p), & p \in P^R \\ \mathbf{offset}'(p) \div s, & p \notin P^R \end{cases}$$

$$p +_p^M i = \begin{cases} p +_p^R i, & (p, i) \in \{(p^R, i^R) \mid p +_p i \in F^*\} \\ (\mathbf{block}^M(p), (\mathbf{address}^M(p) + s \times i)\ \%\ A + 1)_p^M, & (p, i) \notin \{(p^R, i^R) \mid p +_p i \in F^*\} \end{cases}$$

$$p -_p^M q = \begin{cases} p -_p^R q, & (p, q) \in \{(p^R, q^R) \mid p -_p q \in F^*\} \\ (\mathbf{offset}^M(p) - \mathbf{offset}^M(q)) \div s, & (p, q) \notin \{(p^R, i^R) \mid p -_p i \in F^*\} \end{cases}$$

$$p \leq_p^M q = \begin{cases} p \leq_p^R q, & (p, q) \in \{(p^R, q^R) \mid p \leq_p q \in F^*\} \\ \mathbf{offset}^M(p) \leq \mathbf{offset}^M(q), & (p, q) \notin \{(p^R, q^R) \mid p \leq_p q \in F^*\} \end{cases}$$

Рис. 3. Определение восстановленной модели M формулы F в теории BPA

Fig. 3. Definition of the reconstructed model M for the formula F in BPA

Полнота процедуры инстанцирования напрямую следует из перечисленных лемм:

Теорема 1. *Каждая бескванторная формула* F *выполнима в теории IPA тогда и только тогда, когда её трансляция* F^* *выполнима в логике QF_UFLIA.*

6. Формализация

Доказательство полноты, описанное в предыдущем разделе, и было формализовано в системе Isabelle/HOL. С целью обобщения формализации на случай произвольных расширений логики QF_UFLIA различными аксиоматически заданными теориями для формального представления формул мы использовали нетипизированное глубокое погружение всего с двумя конструкторами — применение функции к аргументам (`App`) и подкванторная переменная (`Var`). Свободные переменные, или иначе неинтерпретируемые константы, представляются в этом случае как применения соответствующей неинтерпретируемой

функции к пустому списку аргументов. Кванторы явным образом не включались в представление, вместо этого для каждой формулы из множества аксиом теории неявно использовалась схематическая квантификация по всем подкванторным переменным, которые встречаются в теле аксиомы. Также для определения семантики этих кванторов было определено понятие означивания незамкнутого выражения, в котором вместо оценки свободных (подкванторных, но не связанных) переменных используется подстановка. Более подробно этот прием описан в нашей статье [5], которая использует аналогичный подход к интерпретации формул. Таким образом, в ходе данной работе мы подтвердили, что ранее сделанную формализацию из статьи [5] можно успешно использовать как основу в ходе доказательства полноты новой аксиоматической теории.

Также стоит заметить, что в первоначально сформулированной нами версии триггеров для аксиом $(A_{\mathbb{N}})$ и (A_O) использовалось инстанцирование не всеми указательными термами исходной формулы, а только термами p для подтермов вида $\mathbf{align}(p)$. Однако в ходе верификации выяснилось, что такая процедура трансляции не является полной, в частности, тавтология $\mathbf{address}(p) \neq 0 \Rightarrow p -_p (p +_p 0) = 0$ теории BPA не выводима с помощью такой процедуры. Такая ошибка могла бы в принципе найтись и при тестировании процедуры трансляции, но подбор достаточно полного набора формул для проверки процедуры трансляции с помощью тестов также достаточно сложен на практике. Это отчасти оправдывает применение с целью проверки корректности решающих процедур систем автоматизированного доказательства теорем, таких как Isabelle/HOL.

Соответствующая формализации теория Isabelle называется `TSMT_Pointers_Complete` [17].

7. Дальнейшие исследования

Основными направлениями будущей работы включают разработку некоторых предсказуемых (хотя и неполных) подходов к обработке кванторов, встречающиеся в цели без соответствующих триггеров (стратегии для обработки кванторов внутри существующих решателей очень эффективны, но редко предсказуемы). Еще одно важное направление — формализация и оценка решения, основанного на реализации процедуры для других разрешимых теорий, таких как эффективно разрешимый фрагмент теории бит-векторов — монотонные формулы над равенствами между выражениями с побитовыми операциями.

Аксиомы, формализующие семантику побитовых операций могут иметь общий вид $\forall b_1 \ldots b_n\ l_1 \ldots l_m\ i.\ P(i, f_1(\overline{b_{1..n}}, \overline{l_{1..m}})!!\ g_1(\overline{l_{1..m}}, i), \ldots, f_k(\overline{b_{1..n}}, \overline{l_{1..m}})!!\ g_k(\overline{l_{1..m}}, i))$, где $!!-$ это взятие значения бита (как предикат), а $f_j(\overline{b_{1..n}}, \overline{l_{1..m}})$ и $g_j(\overline{l_{1..m}}, i)$ одинаковые (общие) для всех P, то есть в разных аксиомах одной f_j не может соответствовать разный g_j. То есть каждая функция (операция) от векторов может входить в аксиомы только с одним соответствующим индексом, зависящим от подкванторной переменной. Если целевая формула в теории битовых векторов является монотонным предикатом от равенств между побитовыми выражениями, то проверку существования ее модели можно свести к проверке существования некоторого набора индексов битов, в которых нарушены некоторые из равенств. Тогда все аксиомы можно инстанцировать соответствующими сколемовскими индексами, при этом для оставшихся индексов существование модели достаточно будет проверить один раз в общем случае, так как разные биты одной операции не зависят друг от друга в силу ограничения на однократное использование индексов. Побитовыми операциями являются, например, операции из языка программирования Си — `&`, `|`, `^`, `~`, `<<`, `>>` и целочисленные приведения типов. В саму теорию входят эти операции и еще операция взятия бита `!!`. Например, для операции `<<` нужны две аксиомы: $\forall a\ k,\ \forall i > 0,\ i < n - k \longrightarrow (a \ll k!!\ i \leftrightarrow a!!\ i + k)$ и $\forall a\ k,\ \forall i \geq n - k,\ \neg(a \ll k!!\ i)$.

Аналогичным образом можно сформулировать некоторые другие актуальные теории и их фрагменты, например, фрагмент теории операций со списками.

8. Заключение

В работе представлена теория ограниченной адресной арифметики (с Си-подобным разделением на непересекающиеся блоки памяти), позволяющая реализовать соответствующую решающую процедуру в Isabelle/HOL на основе SMT- решателей. Эта решающая процедура позволяет упростить дедуктивную верификацию программ на языке Си. Разработанная решающая процедура основана на эффективной с точки зрения размера термов трансляции формул теории адресной арифметики в существующую и широко поддерживаемую логику QF_UFLIA.

Список литературы / References

[1]. T. Nipkow, M. Wenzel, and L. C. Paulson. Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer-Verlag, 2002. 240 p.

[2]. Vegard Nossum. [PATCH] firmware: declare __{start,end}_builtin_fw as pointers. Available at: https://www.mail-archive.com/linux-kernel@vger.kernel.org/msg1176758.html.

[3]. Xi Wang, Haogang Chen et al. Undefined behavior: What happened to my code? In Proc. of the Asia-Pacific Workshop on Systems, 2012, Article No.: 9.

[4]. Chad R. Dougherty and Robert C. Seacord. C compilers may silently discard some wraparound checks. Available at: https://www.kb.cert.org/vuls/id/162289.

[5]. Р.Ф. Садыков, М.У. Мандрыкин. Верифицированная тактика Isabelle/HOL для теории ограниченных целых на основе инстанцирования и SMT. Труды ИСП РАН, том 32, вып. 2, 2020 г., стр. 107-124 / R. Sadykov, M. Mandrykin. Verified Isabelle/HOL tactic for the theory of bounded integers based on quantifier instantiation and SMT. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 2, 2020. pp. 107-124 (in Russian). DOI: 10.15514/ISPRAS–2020–32(2)–9.

[6]. H. Tuch. Formal memory models for verifying C systems code. PhD Thesis. School of Computer Science and Engineering, The University of New South Wales, Australia, 2008, 249 p.

[7]. Sandrine Blazy and Xavier Leroy. Formal Verification of a Memory Model for C-Like Imperative Languages. Lecture Notes in Computer Science, vol. 3785, 2005, pp. 280-299.

[8]. Y. Moy. Automatic Modular Static Safety Checking for C Programs. PhD Thesis, Université de Paris-Sud 11, Paris, France, 2009, 249 p.

[9]. Jeehoon Kang, Chung-Kil Hur et al. 2015. A formal C memory model supporting integer-pointer casts. ACM SIGPLAN Notices, vol. 50, issue 6, 2015, pp. 326–335.

[10]. K. Memarian and P. Sewell. Clarifying the C memory object model. Available at: http://www.open-std.org/JTC1/SC22/WG14/www/docs/n2012.htm.

[11]. G. Klein, K. Elphinstone et al. seL4: Formal verification of an OS kernel. In Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 207-220.

[12]. N. Bjørner, A.Blass et al. Modular difference logic is hard. Tech. Rep. MSR-TR-2008-140, Microsoft, 2008. Available at: https://www.microsoft.com/en-us/research/publication/modular-difference-logic-is-hard/.

[13]. C. Barrett, P. Fontaine, and C. Tinelli. The SMT-LIB Standard: Version 2.6. Available at: https://smtlib.cs.uiowa.edu/papers/smt-lib-reference-v2.6-r2017-07-18.pdf.

[14]. S. Böhme. Proof reconstruction for Z3 in Isabelle/HOL. In Proc. of the 7th International Workshop on Satisfiability Modulo Theories (SMT '09), 2009, pp. 1-11.

[15]. S. Böhme and T. Weber. Fast LCF-style proof reconstruction for Z3. Lecture Notes in Computer Science, vol. 6172, 2010, pp. 179–194.

[16]. J. Dawson. Isabelle theories for machine words. Electronic Notes in Theoretical Computer Science, vol. 250, no. 1, pp. 55-70, 2009.

[17]. R. Sadykov and M. Mandrykin. Complete decision procedure for the bounded theory of pointer arithmetic based on quantifier instantiation and SMT. Available at: https://forge.ispras.ru/projects/tsmt/repository/, 2021.

Информация об авторах / Information about authors

Рафаэль Фаритович САДЫКОВ – стажер-исследователь в ИСП РАН, аспирант кафедры МаТИС механико-математического факультета МГУ им. М. В. Ломоносова. Сфера научных интересов: верификация программ, теория распределенных вычислений, теория графов, теория автоматов, математическая логика.

Rafael Faritovich SADYKOV – intern researcher of ISP RAS, PhD student of Faculty of Mechanics and Mathematics, Moscow State University. Research interests: program verification, distributed computing theory, graph theory, automata theory, mathematical logic.

Михаил Усамович МАНДРЫКИН – младший научный сотрудник ИСП РАН, кандидат физико-математических наук по специальности «математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей». Сфера научных интересов: формальные методы верификации программ, математическая логика, функциональное программирование, системы типов в языках программирования.

Mikhail Usamovich MANDRYKIN – researcher at ISP RAS, PhD. Research interests: formal methods, program verification, mathematical logic, functional programming, type systems.