DOI: 10.15514/ISPRAS–2020–32(5)–1

Экспертная оценка результатов верификации инструментов верификации моделей программ

В.А. Гратинский, ORCID: 0000-0001-8598-1298 <gratinskiy@ispras.ru>
Е.М. Новиков, ORCID: 0000-0003-3586-3140 <novikov@ispras.ru>
И.С. Захаров, ORCID: 0000-0001-5713-5887 <ilja.zakharov@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. При проверке программ на соответствие спецификациям требований инструменты верификации моделей программ могут выдавать различные результаты верификации. Среди них вердикты, свидетельства нарушений и отчеты о покрытии кода представляют собой наибольшую ценность с точки зрения экспертов, которые хотят обнаружить ошибки в программах и оценить полноту проверки. Для индустриального использования инструментов верификации моделей программ, когда проверяется множество различных требований к различным версиям и конфигурациям многих программ, экспертам необходимы удобные средства автоматизации оценки результатов верификации. В статье рассматриваются соответствующие подходы и их реализация в системе верификации Klever.

Ключевые слова: верификация моделей программ; результат верификации; свидетельство нарушения; покрытие кода; экспертная оценка

Для цитирования: Гратинский В.А., Новиков Е.М., Захаров И.С. Экспертная оценка результатов верификации инструментов верификации моделей программ. Труды ИСП РАН, том 32, вып. 5, 2020 г., стр. 7-20. DOI: 10.15514/ISPRAS–2020–32(5)–1

Expert Assessment of Verification Tool Results

V.A. Gratinskiy, ORCID: 0000-0001-8598-1298 <gratinskiy@ispras.ru>
E.M. Novikov, ORCID: 0000-0003-3586-3140 <novikov@ispras.ru>
I.S. Zakharov, ORCID: 0000-0001-5713-5887 <ilja.zakharov@ispras.ru>
Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Verification tools can produce various kinds of results while checking programs against requirement specifications. Experts, who seek for errors and estimate completeness of verification, mostly appreciate verdicts, violation witnesses and code coverage reports. They need convenient tools for automating the assessment of verification results to apply verification tools in practice when many program configurations and versions are checked against various requirements. In this paper, we propose new methods for expert evaluation of verification results, covering all those problems that are most significant in accordance with our experience in verifying large programs for compliance with a large number of requirements specifications. Some ideas are borrowed from the areas of testing and static analysis. However, specific methods and technical solutions are unique, since the verification results provided by verification tools are either not found in other areas or have special semantics. The paper presents our approaches and their implementation in the Klever software verification framework.

Keywords: software model checking; verification result; violation witness; code coverage; expert assessment

For citation: Gratinskiy V.A., Novikov E.M., Zakharov I.S. Expert Assessment of Verification Tool Results. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 5, 2020, pp. 7-20 (in Russian). DOI: 10.15514/ISPRAS–2020–32(5)–1

1. Введение

Инструменты верификации моделей программ (далее для краткости – *инструменты верификации*) позволяют проверять программы на соответствие набору спецификаций требований автоматизированным и тщательным образом [1]. На вход инструменты верификации принимают *верификационные задачи*. Их формат подробно описан на сайте международных соревнований инструментов верификации (*SV-COMP*) [2, 3]. Каждая верификационная задача должна содержать подготовленный для верификации фрагмент исходного кода целевой программы среднего размера и спецификацию свойств безопасности/живучести, например, достижимость вызова специальной функции или безопасность работы с памятью. Для получения результатов верификации приемлемого качества при рассмотрении программ по частям, а также для проверки большинства реальных требований необходимо дополнить фрагменты программ достаточно точными моделями их окружения.

В этой статье рассматривается верификация больших, быстро развивающихся программ на языке Си, например, ядер операционных систем. Эти программы должны быть декомпозированы на множество фрагментов (десятки, сотни или тысячи), каждый из которых проверяется независимо. Кроме того, из-за некоторых ограничений инструментов верификации также необходимо проверять различные требования к каждому фрагменту программы независимым образом. В результате могут получиться большие наборы верификационных задач, для каждой из которых инструмент верификации выдает результат ее решения. Вопросы получения наборов верификационных задач и выполнения их решения рассматриваются в работе [4].

В соответствии с правилами *SV-COMP* инструмент верификации предоставляет следующие артефакты в качестве результатов решения верификационной задачи (ниже мы называем их *результатами верификации*).

- *Вердикт*, который отвечает на вопрос: «удовлетворяет ли программа спецификации?». Возможны следующие вердикты:
  - *TRUE* – программа корректна с точки зрения инструмента верификации (однако ошибки могут быть пропущены, например, из-за неточных моделей окружения);
  - *FALSE* – программа содержит ошибку (однако это может быть ложное сообщение об ошибке, например, из-за неточного анализа инструмента верификации);
  - *UNKNOWN* – данный вердикт выдается, когда для завершения проверки инструменту верификации не хватает отведенных вычислительных ресурсов или он завершается аварийно из-за внутренних ошибок.
- *Свидетельство нарушения* (*violation witness*) – машиночитаемый файл, содержащий фрагменты формального доказательства обнаруженного нарушения спецификации проверяемых свойств. Инструмент верификации должен выдавать его для вердикта *FALSE* так, чтобы инструменты валидации свидетельств могли бы подтвердить или опровергнуть корректность данного доказательства автоматически [5].
- *Свидетельство корректности* (*correctness witness*) – аналогично для вердикта *TRUE* [6]. Мы не рассматриваем свидетельства корректности в статье, так как они практически не могут помочь экспертам понять соответствующие доказательства корректности.

Кроме того, инструменты верификации могут выдавать дополнительные результаты верификации, которые могут быть полезны для экспертов. Например, CPAchecker выдает следующие файлы:

- *Отчет о покрытии кода*, демонстрирующий исходный код, который считается достижимым с точки зрения инструмента верификации во время проверки.
- *Лог*, который может включать сведения о проводимом анализе, а также сведения о внутренних ошибках инструмента верификации.

Исследователи и разработчики инструментов статического анализа и тестирования добились больших успехов в представлении результатов. В области верификации моделей программ это направление пока развито не так хорошо. Такие работы, как визуализация трасс ошибок в SDV [7] или свидетельств нарушения в рамках *SV-COMP* [8] не предоставляют пользователям достаточно полный набор средств анализа результатов верификации. В данной работе предложены новые методы экспертной оценки результатов верификации, охватывающие все те проблемы, которые являются наиболее существенными в соответствии с нашим опытом верификации больших программ на соответствие большому количеству спецификаций требований. Некоторые идеи заимствованы из областей тестирования и статического анализа. Однако конкретные методы и технические решения уникальны, поскольку результаты верификации, предоставляемые инструментами верификации, либо не встречаются в других областях, либо обладают особенной семантикой.

В следующих разделах подробно рассматриваются предлагаемые методы экспертной оценки результатов верификации и их реализация в системе верификации Klever [4].

2. Методы экспертной оценки результатов верификации

2.1 Вспомогательная терминология

Перед описанием предлагаемых методов необходимо определить несколько вспомогательных терминов:

- *База сборки* – директория, которая генерируется в результате работы Clade [9] и содержит следующие данные в структурированном виде:
  - файлы с исходным кодом целевой программы;
  - сведения о сущностях программы, таких как типы, функции и переменные, а также об их взаимосвязях;
  - информация о процессе сборки программы.
- *Верификационное задание* – набор файлов, включающий базу сборки, спецификации и конфигурационные файлы, необходимые для запуска верификации. Верификационные задания могут быть созданы на основе предустановленных верификационных заданий, соответствующих различным проектам. Пользователи могут создавать несколько версий для каждого верификационного задания, например, при разработке новой спецификации требований. Проекты, предустановленные верификационные задания, верификационные задания и их версии хранятся и отображаются структурированным образом подобно файлам на диске.
- *Верификационная задача* – набор файлов для запуска инструмента верификации. В целом это определение соответствует определению *SV-COMP*, за исключением того, что здесь и далее в состав верификационных задач также включаются конкретное название инструмента верификации, а также специфичные для него опции командной строки и ограничения вычислительных ресурсов. Верификационные задачи генерируются и решаются по мере решения верификационных заданий.
- *Верификационные отчеты* – отчеты о результатах верификации, а также вспомогательные отчеты, которые создаются в ходе решения верификационного задания. Верификационные отчеты могут включать в себя *атрибуты*, характеризующие проверяемые программы и требования, например, названия и версии проектов, названия

фрагментов программ и идентификаторы спецификаций требований.

- *Оригинальные файлы с исходным кодом* – файлы с исходным кодом целевой программы из базы сборки.
- *Дополнительные файлы с исходным кодом* или *модели* – файлы с исходным кодом, которые генерируются на основе спецификаций при решении верификационных заданий. Модели включаются в верификационные задачи наряду с оригинальными файлами с исходным кодом.
- *CIL файлы* – файлы, объединяющие оригинальные и дополнительные файлы с исходным кодом. Эти файлы получаются с помощью инструмента CIL [10]. Каждая верификационная задача включает в себя ровно один CIL файл, так как большинство инструментов верификации не поддерживает никакого другого представления целевой программы. CIL файлы ссылаются на оригинальные и дополнительные файлы с исходным кодом посредством директив препроцессора *#line*.

2.2 Предлагаемые методы

Первоочередная цель предлагаемых методов заключается в том, чтобы представить экспертам результаты верификации удобным и интуитивным образом с учетом того, что их основные задачи таковы.

- Понять, соответствуют ли выдаваемые предупреждения ошибкам (кроме того, необходимо понять первопричины этих ошибок) или ложным сообщениям об ошибках.
- Оценить полноту проверки. Для этого необходимо предоставить экспертам оценку достижимого исходного кода, чтобы они могли убедиться в том, что анализ не прекратился в начале или в середине программы.

Наши основные предложения перечислены ниже.

- Поддержать нескольких пользователей, каждый из которых может иметь определенный набор прав как глобально, так и для конкретных верификационных заданий. Это может быть полезно, потому что несколько человек могут проводить экспертизу одновременно, а некоторым экспертам нельзя доверять оценку важных результатов верификации. Администратор должен иметь возможность назначать глобальные роли новым пользователям. Пользователи, обладающие правами на верификационные задания, должны иметь возможность назначать права на экспертную оценку соответствующих результатов верификации непривилегированным пользователям.
- Связать свидетельства нарушений и покрытие кода с оригинальными файлами с исходным кодом и моделями; выделить те части свидетельств нарушений, которые соответствуют моделям и изменениям связанных с ошибками данных; скрыть части свидетельств нарушений, которые наименее релевантны обнаруженным предупреждениям. Например, мы предлагаем максимально скрывать от экспертов те части свидетельств нарушений, которые соответствуют реализациям моделей, поскольку соответствующие детали обычно не представляют интереса. Однако должна быть возможность анализировать все подробно, поскольку неточные модели могут вызывать ложные сообщения об ошибках и экспертам может потребоваться их изучить.
- Должна быть возможность создавать *экспертные оценки* для свидетельств нарушений. Для каждого предупреждения эксперт должен иметь возможность указать, соответствует ли оно ошибке или ложному сообщению об ошибке, добавить один или несколько тегов и произвольное текстовое описание. Для изменений экспертных отметок должна быть возможность оставлять комментарии, описывающие эти изменения.
- Автоматически ассоциировать экспертные оценки с аналогичными свидетельствами нарушений. Автоматическая оценка может сэкономить достаточно много времени за счет

отсутствия повторного анализа результатов верификации, полученных, например, для разных версий или конфигураций одной и той же программы. Эксперты должны иметь возможность подтвердить или отклонить автоматически ассоциированные оценки вручную, так как автоматическая процедура может работать некорректно.

- Предоставить экспертам статистические данные о вынесенных вердиктах, а также об ассоциированных экспертных оценках. Кроме того, эксперты должны иметь возможность изучить соответствующие списки, например, список всех предупреждений, которые были автоматически оценены как ошибки.
- Должна быть возможность сравнивать результаты верификации друг с другом. Это полезно для оценки как изменений в проверяемых программах, включая сравнение их различных конфигураций, так и изменений в компонентах и спецификациях Klever.
- Пользователи должны иметь возможность настраивать представление результатов верификации, чтобы анализировать их более удобным способом, например, добавлять или удалять некоторые столбцы в таблицах, сортировать и фильтровать результаты верификации по некоторым критериям.
- Позволить перемещать результаты верификации, а также экспертные оценки между различными машинами. Это необходимо, поскольку результаты верификации могут быть получены в разное время на разных машинах, включая те, которые не соединены по сети. Однако может оказаться более удобным хранить и анализировать все результаты верификации на одной машине.
- Сохранять как версии верификационных заданий, так и полученные для них результаты верификации. Например, это может быть полезно для сравнения их друг с другом через некоторое время.
- Сохранять историю изменений экспертных оценок для того, чтобы иметь возможность откатить некоторые изменения и проанализировать эту историю.
- Должна быть возможность проведения экспертной оценки результатов верификации по мере их получения, так как этот процесс может занять значительное количество времени, например, несколько дней даже на достаточно мощной машине.

Реализация предлагаемых методов должна быть достаточно эффективной и не создавать значительную нагрузку на диск и память. Для этого необходимо учесть следующее.

- Нельзя выполнять сложные вычисления непосредственно при представлении результатов верификации пользователям. Например, данные для представления перекрестных ссылок должны быть получены до того, как будут визуализироваться соответствующие файлы с исходным кодом.
- Необходимо кэшировать данные, которые нетривиально получить и которые невозможно вычислить заранее. Например, это касается сравнения результатов верификации.
- Нельзя дублировать оригинальные файлы с исходным кодом и вспомогательную информацию для них, такую как данные для подсветки синтаксиса и перекрестных ссылок. Благодаря этому можно существенно уменьшить занимаемое дисковое пространство (для больших программ, таких как ядро Linux, соответствующие данные могут занимать сотни мегабайт даже в архивах). Как правило, это требование не актуально для моделей, поскольку они довольно часто меняются. Кроме того, большинство моделей имеют несколько десятков или сотен строк.
- Должны существовать средства для удаления ненужных данных, таких как промежуточные версии верификационных заданий и их результаты верификации. Это может помочь сократить объем отображаемой информации и очистить диск.

Наконец, мы предлагаем поддержать специальные возможности для разработчиков.

- Показ логов работы компонентов Klever.
- Поддержка автоматизированной оценки верификационных отчетов с информацией о внутренних ошибках компонентов Klever и инструментов верификации.
- Загрузка архивов конкретных верификационных задач. Например, это может быть полезно для того, чтобы подготовить для разработчиков инструментов верификации примеры, когда инструменты верификации работают некорректным образом.

В следующих подразделах подробно рассматриваются наиболее важные и сложные подходы, предложенные выше.

2.3 Представление вердиктов

Мы предлагаем рассматривать вердикты *TRUE* и *FALSE*, соответствующие доказательствам корректности и обнаружению нарушений проверяемых спецификаций требований, как *Safe* и *Unsafe* соответственно, так как это должно быть более понятно экспертам. Если при решении одной верификационной задачи обнаружено более одного нарушения проверяемой спецификации требований, то должно быть создано соответствующее количество верификационных отчетов *Unsafes*, а каждый из них должен быть дополнен своим собственным свидетельством нарушения.

2.4 Представление свидетельств нарушений и их экспертная оценка

2.4.1 Формат свидетельств нарушений

Свидетельство нарушения – это специальный XML-файл в формате *graphML* [5, 6, 11]. Этот файл описывает наблюдающий автомат (*observer automaton*) для обхода программы от точки входа до найденной ошибки. Инструмент валидации свидетельств рассматривает наблюдающий автомат в сочетании с автоматом потока управления, извлекаемым из представления программы, для проверки корректности решения верификационной задачи. Свидетельство нарушения не является трассой ошибки, поскольку оно может описывать подмножество возможных путей выполнения и может не содержать нужных деталей, таких как части путей через некоторые функции, итерации цикла и т.д.

Далее в данном подразделе рассматриваются ключевые элементы формата свидетельств нарушений. Пути выполнения программы задаются с помощью вершин и ребер. Промежуточные вершины имеют входные и выходные ребра. Тупиковые вершины (*sink nodes*) не имеют выходных ребер. Пример тупиковой вершины – это вершина, соответствующая найденной ошибке.

Каждое ребро соответствует некоторой декларации или выражению в программе. Оно ссылается на них с помощью номера строки и смещения в CIL файле (теги *startline*, *endline*, *startoffset* и *endoffset*). Для указания входа и выходы из функции, условного выражения и цикла используются теги *enterFunction*, *returnFromFunction*, *control* и *enterLoopHead* соответственно. Для представления значений переменных, которые предполагает инструмент верификации, используется тег *assumption*.

На Листингах 1 и 2 приведены примеры программы на языке Си, содержащей ошибку, и соответствующего свидетельства нарушения соответственно. Для краткости для свидетельства нарушения не показаны некоторые теги, тупиковые вершины, а также ребра, ведущие к ним.

Данный пример демонстрирует ситуацию, когда свидетельство нарушения не содержит важный фрагмент пути выполнения через функцию *func* (ребро от вершины *A2* до вершины *A3*). Хотя такое свидетельство нарушения корректно и пригодно для валидации, оно не может использоваться в качестве полноценной трассы ошибки.

```
 1 int func(int arg) {
 2     return arg > 0 ? 1 : 0;
 3 }
 4 void entry_point(int arg) {
 5     usigned int cond = 0;
 6     if (arg)
 7         cond = func(arg);
 8     if (arg)
 9         assert(cond != 1);
10 }
```

Листинг 1. Пример программы на языке Си, содержащей ошибку
Listing 1. An example of a C program containing an error

```
<node id="A0">
    <data key="entry">true</data>
</node>
<edge source="A0" target="A1">
    <data key="startline">4</data>
</edge>
<node id="A1"/>
<edge source="A1" target="A2">
    <data key="startline">6</data>
    <data key="control">condition-true</data>
    <data key="assumption">arg == 1;</data>
</edge>
<node id="A2"/>
<edge source="A2" target="A3">
    <data key="startline">7</data>
</edge>
<node id="A3"/>
<edge source="A3" target="A4">
    <data key="startline">8</data>
    <data key="control">condition-true</data>
    <data key="assumption">cond == 1;</data>
</edge>
<node id="A4">
    <data key="violation">true</data>
</node>
```

Листинг 2. Пример свидетельства нарушения
Listing 2. An example of a violation witness

Для проверки большинства требований (свойств) верификационные задачи должны включать исходный код целевых программ и моделей окружения, которые сериализуют возможные выполнения каким-либо разумным образом, поскольку тщательная проверка многопоточных программ является чрезвычайно сложной задачей. Тем не менее, например, для поиска состояний гонок инструменты верификации должны получать многопоточные программы и модели окружения в качестве входных данных. В этом случае они добавляют для ребер свидетельств нарушений информацию о потоках, в которых они выполняются, в частности идентификаторы потоков.

2.4.2 Расширенный формат свидетельств нарушений

Мы предлагаем расширить формат свидетельств нарушения таким образом, чтобы иметь все части пути [12]. Кроме того, данный формат позволяет добавить важную информацию от алгоритмов проверки для ребер свидетельств нарушения. Благодаря этому, например, при проверке безопасной работы с памятью можно отметить те места, где выделяется память, которая впоследствии не освобождается (утечка памяти).

2.4.3 Преобразование свидетельств нарушений в трассы ошибок

В данной работе предлагается преобразовывать свидетельства нарушений в трассы ошибок, которые более удобны для визуализации и анализа [13]. Данное преобразование включает в себя следующие действия.

- Преобразование ссылок на CIL файлы для ребер свидетельств нарушений в соответствующие декларации и выражения, а также номера строк в оригинальных и дополнительных файлах с исходным кодом программы.
- Обработка специальных комментариев из моделей окружения и спецификаций требований для определения действий со специальными сущностями, таких как вызовы точек входа фрагментов программы и изменения состояния модели. В соответствии с этими данными все части свидетельств нарушения, которые кажутся несущественными для обнаруженных нарушений проверяемых требований, помечаются так, чтобы их можно было скрыть при последующей визуализации.
- Если инструмент верификации не предоставляет информацию о потоках для ребер, мы предлагаем добавить ее в соответствии с комментариями в моделях окружения. Такие комментарии указывают на создание и объединение потоков, а автоматически генерируемые идентификаторы потоков распространяются по соответствующим стекам вызовов.

2.4.4 Представление покрытия кода

Отчеты о покрытии кода являются важным артефактом при применении верификации на практике. Они отражают части программы, такие как строки кода, ветви и функции, которые фактически проверяются. Эта информация необходима для оценки качества модели окружения, поскольку ни свидетельства нарушений, ни свидетельства корректности не включают в себя данные о рассматриваемых путях. Отчеты о покрытии кода помогают понять, какие точки входа в программу должны дополнительно вызываться из моделей окружения.

Отчет о покрытии кода – это вспомогательный файл, описывающий исходный код верификационной задачи, который был достижим во время верификации. Этот файл предназначен только для визуализации пользователям. Насколько известно авторам статьи, CPAchecker – это единственный инструмент верификации, который может выдавать отчеты о покрытии кода в дополнение к свидетельствам [14]. Формат файла соответствует формату тестового покрытия *LCOV* [15].

Как и для свидетельств нарушений, мы предлагаем преобразовать отчеты о покрытии кода в более подходящую форму для их визуализации [16], что включает в себя следующие шаги.

- Связывание покрытия кода с соответствующими оригинальными файлами с исходным кодом и моделями вместо CIL файлов.
- Вычисление статистики для каждого отдельного файла с исходным кодом, а также для каждой поддиректории дерева исходного кода программы с учетом либо файлов с исходным кодом, рассматриваемых инструментом верификации, либо всех файлов с исходным кодом из базы сборки. Эти наборы файлов могут отличаться, так как инструмент верификации может анализировать не все файлы с исходным кодом,

например, из-за его внутренних ошибок.

- Объединение отчетов о покрытии кода, выданных для отдельных верификационных задач, с целью получить общий отчет о покрытии кода для программы независимо для каждой спецификации требований.

2.4.5 Сравнение результатов верификации

При сравнении результатов верификации двух версий одного или двух различных верификационных заданий предлагается либо получить данные по сравнению из кэша, либо вычислить их и заполнить кэш в соответствие со следующим алгоритмом:

- Для каждой из двух версий верификационного(ых) задания(й) определить список атрибутов, которые должны использоваться для сравнения, в соответствии со значениями специального флага атрибутов верификационных отчетов. Упорядочить полученные списки по именам атрибутов в алфавитном порядке. Типичный список атрибутов состоит из имени фрагмента программы и идентификатора спецификации требований, так как обычно они однозначно различают верификационные задачи верификационных заданий.
- Сообщить, что верификационные результаты не могут быть сравнены, если полученные списки атрибутов не совпадают. В противном случае перейти к следующим шагам.
- Получить вердикты для всех верификационных отчетов, соответствующих уникальным значениям атрибутов из списка, и вычислить суммарный вердикт для каждого такого списка верификационных отчетов.
- Для каждого уникального набора значений атрибутов сохранить в кэше следующее: набор значений атрибутов, суммарный вердикт первой и второй версии верификационного(ых) задания(й). Когда список отчетов одной из версии верификационного задания пуст, необходимо сохранить специальный суммарный вердикт *Unmatched*.
- Для каждой пары суммарных вердиктов рассчитать количество соответствующих верификационных отчетов.

На основе данных в кэше предлагается показать количество верификационных отчетов для всех возможных пар суммарных вердиктов. Кроме того, эксперты должны иметь возможность анализировать конкретные верификационные отчеты, соответствующие уникальным значениям атрибутов, а также ассоциированные с ними экспертные оценки, если таковые имеются.

3. Реализация предложенных методов

Предложенные методы экспертной оценки результатов верификации были реализованы в компоненте Klever Bridge системы верификации Klever [4, 17]. Оценить удобство и производительность Klever Bridge довольно сложно, поскольку существует множество различных вариантов использования, некоторые из которых предполагают интенсивное взаимодействие с пользователем. Хотя некоторые действия, например, анализ и экспертная оценка свидетельств нарушений, не являются тривиальными и широко известными, как правило, новые пользователи начинают выполнять данные действия самостоятельно после нескольких часов обучения.

Для демонстрации Klever Bridge мы перепроверили безопасность работы с памятью для всех загружаемых модулей ядра Linux 5.5 (архитектура *x86_64* и конфигурация *allmodconfig*) на новом инстансе Klever и загрузили на него экспертные оценки, созданные на старом инстансе Klever.

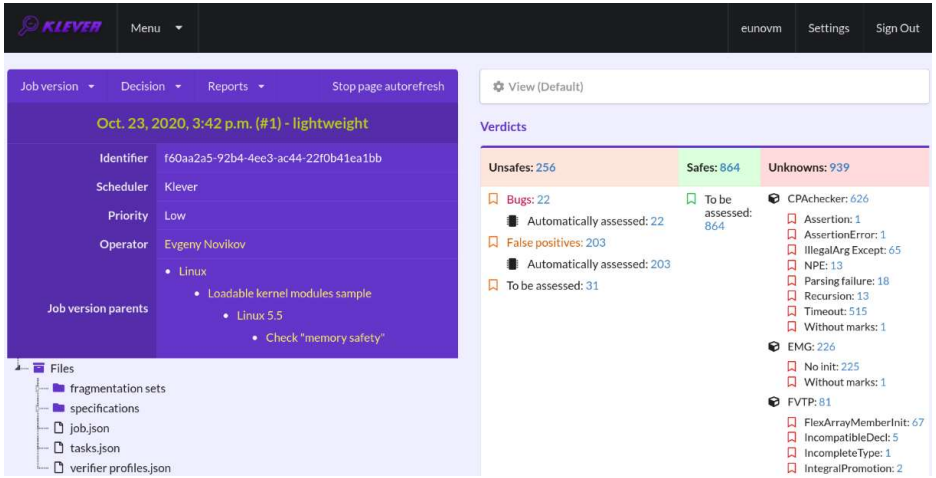

Рис. 1. Статистика по вердиктам и ассоциированным экспертным оценкам
Fig. 1. Statistics on verdicts and associated expert marks

Рис. 1 показывает статистику по вердиктам и ассоциированным экспертным оценкам. Можно увидеть, что большинство *Unsafes* были оценены автоматически. На рис. 2 представлен пример визуализации свидетельства нарушения, файлов с исходным кодом и покрытия кода для загружаемого модуля ядра (фрагмента программы) *drivers/media/platform/s5p-jpeg/s5p-jpeg.ko*. Выданное предупреждение соответствует ошибке в том случае, если функция *of_match_node()* может завершиться не успешно. В этом случае *jpeg_get_drv_data()* возвращает *NULL*, который присваивается указателю *jpeg→variant*, разыменовываемому позже без каких-либо проверок. Исправление данной ошибки были сообщено разработчикам ядра Linux [18].

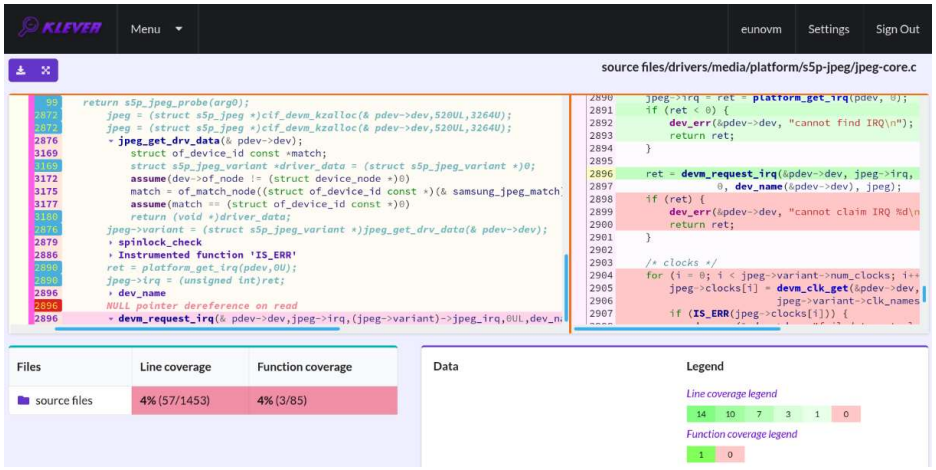

Рис. 2. Визуализация свидетельства нарушения, файлов с исходным кодом и покрытия кода
Fig. 2. Visualized violation witness, source files and code coverage

Автоматически ассоциированные экспертные оценки для данного свидетельства нарушения представлены на рис. 3. Вторая экспертная оценка в списке была создана для старой версии спецификаций, таким образом, она не похожа на новое свидетельство нарушения. Первая

экспертная оценка ассоциирована со свидетельством нарушения на 100%, и эксперт может ее подтвердить.

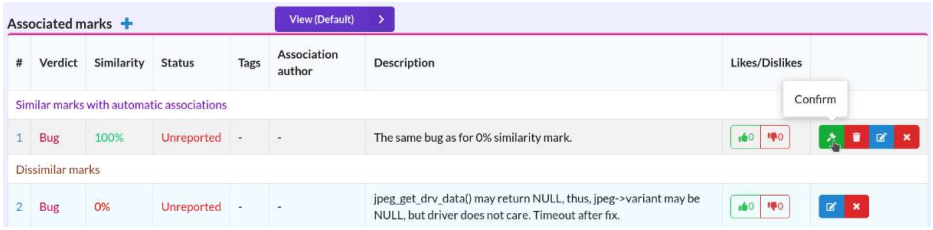

Рис. 3. Автоматически ассоциированные экспертные оценки
Fig. 3. Automatically associated expert marks

Рис. 4 демонстрирует сравнение тех результатов верификации, которые рассматривались выше, с результатами верификации, полученными для Linux 5.9-rc8. Большинство результатов верификации остались теми же самыми, однако есть и много различий. Например, под таблицей приведены детали для отличия, которое произошло после того, как мы отправили патч разработчикам ядра Linux.

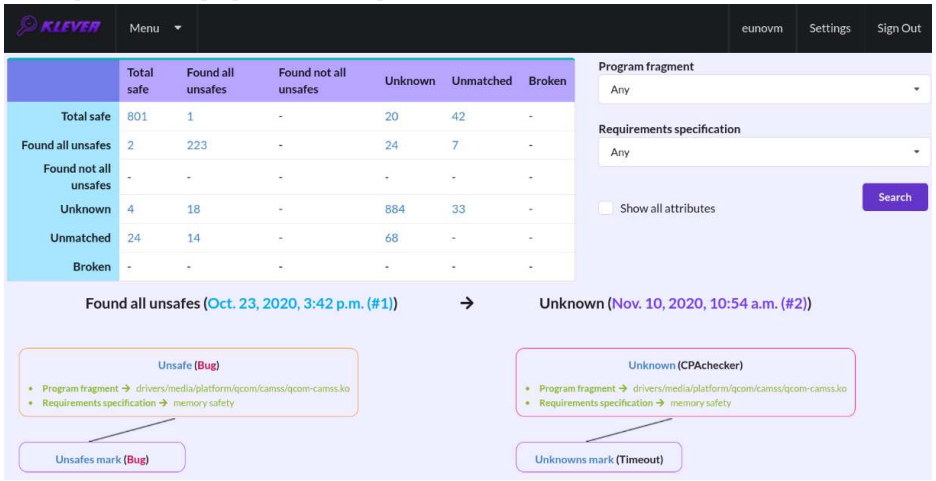

Рис. 4. Сравнение результатов верификации
Fig. 4. Comparison of verification results

Производительность Klever Bridge достаточно высокая: несколько пользователей могут выполнять большинство доступных операций с десятками больших верификационных заданий и сотнями экспертных оценок без существенных задержек. При этом нагрузка на память и на диск незначительна.

Расширенный формат свидетельств нарушений был реализован в инструменте верификации CPAchecker. Другие инструменты верификации не выдают свидетельства нарушений в данном формате, поэтому их визуализация и экспертная оценка могут быть не настолько наглядны и эффективны, как для CPAchecker.

4. Заключение

Опыт практического применения инструментов верификации моделей для больших, быстро развивающихся программ показывает, что недостаточно подготовить верификационные задачи, запустить инструменты верификации и получить результаты верификации, такие как

вердикты, свидетельства нарушений и отчеты о покрытии кода. Экспертам нужны удобные и эффективные инструменты для их хранения, визуализации и анализа.

В данной статье рассмотрены существующие подходы и предложены новые методы экспертной оценки результатов верификации. Предложенные методы были реализованы в системе верификации Klever, которая достаточно успешно используется при проверке различного программного обеспечения. На странице проекта [19] можно найти список возможных улучшений Klever в рассматриваемом направлении. Кроме того, у нас есть планы транслировать свидетельства нарушений других инструментов верификации с помощью CPAchecker, чтобы повысить наглядность их визуализации, а также поддержать их автоматизированную экспертную оценку.

Список литературы / References

[1]. R. Jhala and R. Majumdar. Software model checking. ACM Computing Surveys, vol. 41, issue 4, 2009, article 21, 54 p.
[2]. D. Beyer. Advances in Automatic Software Verification: SV-COMP 2020". Lecture Notes in Computer Science, vol. 12079, 2020, pp. 347-367.
[3]. Definitions and Rules of the 10th International Competition on Software Verification. URL: https://sv-comp.sosy-lab.org/2021/rules.php, accessed 27.11.2020.
[4]. E. Novikov and I. Zakharov. Towards automated static verification of GNU C programs. Lecture Notes in Computer Science, vol. 10742, 2018, pp. 402-416.
[5]. D. Beyer, M. Dangl, D. Dietsch, M. Heizmann, and A. Stahlbauer. Witness validation and stepwise testification across software verifiers. In Proc. of the 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 721-733.
[6]. D. Beyer, M. Dangl, D. Dietsch, and M. Heizmann. Correctness witnesses: exchanging verification results between verifiers. In Proc. of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 326-337.
[7]. Static Driver Verifier Report. URL: https://docs.microsoft.com/en-us/windows-hardware/drivers/devtest/static-driver-verifier-report, accessed 27.11.2020.
[8]. D. Beyer and M. Dangl. Verification-aided debugging: An interactive web-service for exploring error witnesses. Lecture Notes in Computer Science, vol. 9780, 2016, pp. 502-509.
[9]. Clade. URL: https://github.com/17451k/clade, accessed 27.11.2020.
[10]. G.C. Necula, S. McPeak, S.P. Rahul, W. Weimer. CIL: Intermediate Language and Tools for Analysis and Transformation of C Programs. Lecture Notes in Computer Science, vol. 2304, 2002, pp. 213-228.
[11]. Exchange Format for Violation Witnesses and Correctness Witnesses. URL: https://github.com/sosy-lab/sv-witnesses, accessed 27.11.2020.
[12]. Klever extended violation witness format. URL: https://klever.readthedocs.io/en/latest/dev.html#extended-violation-witness-format, accessed 27.11.2020.
[13]. Klever error trace format. URL: https://klever.readthedocs.io/en/latest/dev.html#error-trace-format, accessed 27.11.2020.
[14]. R. Castaño, V. Braberman, D. Garbervetsky, and S. Uchitel. Model checker execution reports. In Proc. of the 32nd IEEE/ACM International Conference on Automated Software Engineering, 2017, pp. 200-205.
[15]. LCOV. URL: https://github.com/linux-test-project/lcov, accessed 27.11.2020.
[16]. Klever code coverage format. URL: https://klever.readthedocs.io/en/latest/dev.html#code-coverage-format, accessed 27.11.2020.
[17]. Klever. URL: https://forge.ispras.ru/projects/klever, accessed 27.11.2020.
[18]. [PATCH] s5p-jpeg: hangle error condition in s5p_jpeg_probe. URL: https://lkml.org/lkml/2020/11/13/673, accessed 27.11.2020.
[19]. Klever Bridge issues. URL: https://tinyurl.com/yyq9ljk8, accessed 27.11.2020.

Информация об авторах / Information about authors

Владимир Анатольевич ГРАТИНСКИЙ – программист отдела технологий программирования. Сфера научных интересов: методы автоматизации анализа и оценки результатов верификации моделей программ с помощью графических интерфейсов.

Vladimir Anatolyevich GRATINSKIY – software developer at the Software Engineering Department. Research interests: methods for automating analysis and evaluation of results of software model checking using graphical interfaces.

Евгений Михайлович НОВИКОВ – кандидат физико-математических наук, старший научный сотрудник и руководитель программных проектов отдела технологий программирования. Его научные интересы включают верификацию моделей программ, спецификацию требований и операционные системы.

Evgeny Mikhailovich NOVIKOV – PhD, Senior Researcher and Software Project Manager at the Software Engineering Department. His research interests include software model checking, requirements specification and operating systems.

Илья Сергеевич ЗАХАРОВ – кандидат физико-математических наук, младший научный сотрудник отдела технологий программирования. Его научные интересы включают методы формальной верификации системного программного обеспечения, в частности операционных систем.

Ilja Sergeevich ZAKHAROV – Ph.D., Junior Researcher at the Software Engineering Department. His research interests include formal verification and model checking of system software and operating systems in particular.