Санкт-Петербургский государственный университет

Суханова Анжела Кирилловна

Выпускная квалификационная работа

Фаззинг решателя Spacer

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2018 «Программная инженерия»*

Научный руководитель:
доцент кафедры информатики, к.ф.-м.н., Григорьев С. В.

Рецензент:
программист-исследователь
АНО ДПО «Научно-Исследовательский и Образовательный Центр «ДжетБрейнс»
Костюков Ю. О.

Санкт-Петербург
2022

Saint Petersburg State University

Anzhela Sukhanova

Bachelor's Thesis

Spacer fuzzing

Education level: bachelor

Speciality *09.03.04 «Software Engineering»*

Programme *CB.5080.2018 «Software Engineering»*

Scientific supervisor:
C.Sc., docent Semyon Grigorev

Reviewer:
Researcher at Research and Education Center JetBrains INO APE Yurii Kostyukov

Saint Petersburg
2022

Оглавление

Введение

Тщательное тестирование и анализ программного обеспечения — основа его безопасности и корректности. Так как в условиях усложнения программных систем «ручная» проверка не может обеспечить требуемый уровень качества, то большой интерес вызывают методики автоматизированного тестирования.

В зависимости от того, исполняется ли проверяемая программа во время тестирования или нет, используемый метод тестирования можно отнести к одному из трёх видов анализа ПО: статическому, динамическому или гибридному анализу, совмещающему в себе первые два. Статический анализ производится без выполнения исследуемых программ, и к нему относятся символьное исполнение (symbolic execution), проверка моделей (model checking), анализ потока данных (data flow analysis). Методами динамического анализа, подразумевающего исполнение проверяемых программ, являются фаззинг (fuzzing), профилирование (profiling) и другие.

Использование в статическом анализе решателей систем дизъюнктов Хорна с ограничениями (Constrained Horn Clauses, CHC) является перспективным направлением [15]. Дизъюнкты Хорна с ограничениями — это формулы определённой структуры в теориях первого порядка, и программы могут быть смоделированы в виде систем таких формул [15]. Существуют эффективные [10] решатели систем дизъюнктов Хорна с ограничениями (далее — Хорн-решатели), т.е. инструменты проверки выполнимости таких систем: Spacer [16], Eldarica [14], PCSat [22] и другие.

Статические анализаторы тоже нуждаются в тестировании. Их проверка особенно важна, так как из-за неточности анализаторов ошибки исследуемых программ могут остаться необнаруженными. В частности, актуальной задачей является проверка используемых ими Хорн-решателей. Техникой автоматического тестирования, зарекомендовавшей себя в проверке инструментов обеспечения безопасности и качества ПО [3, 9, 1, 12], является фаззинг. Фаззинг позволяет быстро проверять

нетривиальные свойства программ и анализировать разные сценарии их исполнения. Инструменты, реализующие данный подход, называются фаззерами.

Решатель систем дизъюнктов Хорна с ограничениями Spacer стабильно лидирует на ежегодном соревновании Хорн-решателей — CHC-COMP [5]. Эта работа посвящена фаззингу решателя Spacer. Поскольку в доступной научной литературе не было найдено информации о фаззерах решателей систем дизъюнктов Хорна, то требуется предложить подход, который фаззер будет реализовывать. Кроме того, в процессе тестирования возникает необходимость сообщать о найденных проблемах разработчикам анализируемой программы. Людям легче понимать и исправлять небольшие ошибки, которые легко воспроизводятся. Поэтому необходимо разработать вспомогательный инструмент, который позволит упростить случаи возникновения ошибок, обнаруженных фаззером.

Разработка фаззера для тестирования решателя Spacer ведётся с июля 2021 года в научно-исследовательской лаборатории верификации и анализа программ при Институте компьютерных наук и технологий Санкт-Петербургского политехнического университета.

1. Постановка задачи

Целью данной работы является разработка фаззера для тестирования решателя Spacer. Для её достижения были поставлены следующие задачи.

- Предложить способ фаззинга Хорн-решателей.
- Спроектировать и создать фаззер, реализующий представленный способ.
- Испытать разработанный инструмент на задаче фаззинга Хорн-решателя Spacer.
- Реализовать инструмент, упрощающий примеры, на которых возникают ошибки.

2. Обзор

В этом разделе дано определение дизъюнктов Хорна с ограничениями и рассмотрены решатели систем таких дизъюнктов. Также в разделе изложено, что из себя представляет фаззинг и какие фаззеры решателей существуют. В завершение обзора описывается метрика эффективности фаззера.

2.1. Дизъюнкты Хорна с ограничениями

Дизъюнкты Хорна с ограничениями — формулы логики первого порядка следующего вида: $\forall V(\phi \wedge p_1(X_1) \wedge \cdots \wedge p_n(X_n)) \rightarrow h(X)$, где

- ϕ — формула в теории первого порядка, называемая ограничением;
- V — переменные;
- $X_1, \ldots X_n$ — термы над переменными V;
- $p_1, \ldots, p_n$ — неинтерпретированные предикатные символы с заданной арностью;
- h — неинтерпретированный предикатный символ с заданной арностью или $\bot$ [15].

Дизъюнкт Хорна с ограничениями называется линейным, если в его посылку входит не более одного неинтерпретированного предикатного символа. Система дизъюнктов линейна, если каждый дизъюнкт в ней линеен. Соответственно, система является нелинейной, если хотя бы один её дизъюнкт нелинеен. Такие системы дизъюнктов Хорна с ограничениями хуже поддаются решению, чем линейные системы [19]. Правилами будем называть дизъюнкты Хорна, содержащие в следствии импликации неинтерпретированный предикатный символ.

2.2. Решатели систем дизъюнктов Хорна

В данной работе рассматриваются Хорн-решатели Spacer и Eldarica. Решатели систем дизъюнктов Хорна используют SMT-решатели для проверки выполнимости ограничений. SMT (satisfiability modulo theories, задача выполнимости формул в теориях) — задача выполнимости формул в логике первого порядка, где функциональные и предикатные символы интерпретируются согласно конкретным теориям. SMT-решатели автоматически определяют выполнимость формул в теориях.

На текущий момент одним из самых эффективных Хорн-решателей является Spacer. Spacer — решатель систем дизъюнктов Хорна с ограничениями, часть системы Z3. Z3 [28] — один из самых эффективных SMT-решателей, и Spacer использует его компоненты для проверки выполнимости. Этот Хорн-решатель поддерживает работу с линейной вещественной и целочисленной арифметикой, теорией массивов, а также предлагает ограниченные возможности для работы с теорией битвекторов, нелинейной арифметикой и другими теориями [13]. Кроме того, Spacer позволяет решать нелинейные дизъюнкты.

Также высокие результаты показывает решатель Eldarica [14].

Eldarica — решатель систем дизъюнктов Хорна с ограничениями с открытым исходным кодом, поддерживающий целочисленную арифметику, теорию массивов, алгебраические типы данных и теорию битвекторов. Eldarica использует SMT-решатель Princess [21].

2.3. Фаззинг и метаморфическое тестирование

Фаззинг — это методика автоматического тестирования ПО, заключающаяся в анализе реакции программы на неожиданные или случайные входные данные [23].

Входные данные для фаззинга могут генерироваться с нуля (например, если для них задана порождающая формальная грамматика) или путём изменения существующих данных [20]. Изменения, которым подвергаются начальные данные при использовании второго способа, принято называть мутациями, а получившиеся в результате изменения

примеры — мутантами.

Одним из методов мутационного фаззинга, позволяющим быстро проверять сложные свойства входных данных, является метаморфическое тестирование [7]. Метаморфическое тестирование — подход, согласно которому данные изменяются таким образом, что на протяжении всего тестирования между начальным примером и его мутантами сохраняется некоторое общее свойство, так называемая метаморфическая связь.

2.4. Фаззеры решателей

В доступной научной литературе не было найдено информации о фаззерах Хорн-решателей, однако существуют фаззеры SMT-решателей. Наиболее известные из них:

- STORM [9] — мутационный black-box[1] фаззер SMT-решателей.
- BanditFuzz [1] — grey-box[2] фаззер SMT-решателей, основанный на машинном обучении. BanditFuzz генерирует тесты по поданной ему на вход грамматике, и мутирует их с сохранением описанной ею структуры. В процессе работы фаззер исследует, какие грамматические конструкции приводят к ошибкам.
- FuzzSMT [3] — black-box фаззер, случайным образом генерирующий по грамматике входную формулу в теории массивов или битовых векторов.
- Falcon [12] — фаззер, генерирующий примеры по грамматике, который основан на изучении функций, используемых SMT-решателями, и их корреляции с сгенерированными входными формулами.

Фаззеры SMT-решателей крайне неэффективны в тестировании Хорн-решателей. Основная проблема состоит в том, что генерируемые фаззе-

[1]Black-box тестирование — техника проверки без доступа к внутреннему коду исследуемой системы.

[2]Grey-box тестирование подразумевает частичный доступ к внутренней структуре проверяемой системы.

рами SMT-решателей примеры обычно не сохраняют структуру дизъюнктов Хорна с ограничениями.

Так, например, STORM за сутки работы на наборе систем дизъюнктов Хорна с ограничениями, сделав множество запусков (то есть мутировав формулу и передав её на решение), ни разу не сохранил структуру систем дизъюнктов Хорна с ограничениями. Это объясняется тем, как STORM генерирует новые примеры. STORM выбирает случайную интерпретацию свободных переменных и рассматривает произвольный набор подформул начальной системы. Далее в соответствии с интерпретацией он составляет пример из выбранных подформул. При таком подходе вероятность получить систему дизъюнктов Хорна крайне мала.

Несмотря на то, что BanditFuzz способен генерировать тесты заданной структуры, он не может быть использован для поиска ошибок построения модели и решения системы. BanditFuzz спроектирован под поиск ошибок производительности. Используемые им мутации не сохраняют выполнимость примеров, то есть к нему не применим подход метаморфического тестирования. Можно было бы проверять генерируемые им примеры разными Хорн-решателями и сравнивать их результаты решения, однако такой подход мало эффективен как основной. Так как Spacer лучше справляется с системами дизъюнктов, чем многие другие решатели, то будут нередки случаи, когда результат его работы не с чем сравнить.

Из-за того, что FuzzSMT и Falcon генерируют примеры с нуля, им сложно создать множество систем дизъюнктов Хорна с ограничениями, покрывающих разные сценарии решения. Кроме того, создавая входные данные таким образом, нельзя предсказать выполнимость порождаемых формул. Это возвращает нас к проблеме проверки формул другим Хорн-решателем.

Также фаззеры SMT-решателей совсем не учитывают ход решения дизъюнктов Хорна и реализаций конкретных решателей. Таким образом, переиспользование существующих фаззеров для решения поставленной задачи практически равнозначно их переписыванию и проигры-

вает созданию фаззера с нуля в удобстве разработки и использования.

2.5. Выбор метрики качества фаззера

Существуют метрики, позволяющие оценить эффективность работы фаззера:

1. покрытие программного кода тестируемой программы [11];
2. количество уникальных трасс исполнения, открытых тестируемой программой [4].

Для фаззинга Хорн-решателей больше подходит вторая метрика, так как интерес представляет именно изменение хода решения, а не покрытие. Имеет значение, сколько раз и когда решателем посещались конкретные строчки кода, так как это может означать, что решение пошло по другому сценарию. Статистика о покрытии такую информацию не предоставляет. Таким образом, в данной работе качество фаззера оценивается с помощью подсчета уникальных трасс исполнения, открытых Хорн-решателем.

3. Проектирование и реализация

В данном разделе описаны основные идеи и архитектура фаззера и используемые им мутации. Кроме того, представлены технические решения и дано описание схемы упрощения примеров, приводящих к ошибкам, а также порождающих их цепочек мутаций.

3.1. Фаззинг решателей систем дизъюнктов Хорна с ограничениями

Так как очень сложно создавать с нуля разнообразные системы дизъюнктов Хорна с ограничениями, не имеющие тривиальных решений, то имеет смысл проектировать именно мутационный фаззер. Кроме того, это облегчается тем, что множество разнообразных систем дизъюнктов Хорна, которые можно использовать как начальные данные, используются на соревнованиях и в научных статьях.

Серьезными ошибками решения дизъюнктов Хорна являются неверный вывод о выполнимости формулы [9], а также некорректное построение модели (если анализируемая формула выполнима). Такое сложное свойство как выполнимость легко проверяется с помощью метаморфического тестирования. Эта методика легла в основу проектируемого фаззера. Метаморфическая связь в данном случае — выполнимость формулы. То есть на каждой итерации работы фаззер применяет к системам дизъюнктов мутации, сохраняющие выполнимость. Если после применения такой мутации результат решения системы отличается от ожидаемого, то обнаружена ошибка (см. таблицу 1). Формулы, выполнимость которых решатель не может определить, фаззер не использует, так как нельзя наверняка сказать, корректно они решались или нет.

Результат решения / Истина	выполнимо (sat)	невыполнимо (unsat)	неизвестно (unknown)
выполнимо	нет ошибки	ошибка	неизвестно
невыполнимо	ошибка	нет ошибки	неизвестно

Таблица 1: Ошибки определения выполнимости.

Для оценки эффективности разрабатываемого фаззера собирается не вся трасса исполнения, а последовательность основных шагов решения систем дизъюнктов Хорна. При полном сборе трассы незначительные изменения, не отражающиеся на процессе решения (например, удаление подвыражений или другие вспомогательные действия с ними) могут приводить к увеличению числа уникальных трасс. То есть при несущественном изменении системы дизъюнктов, значение метрики будет увеличиваться. Таким образом, имеет смысл только выборочный сбор этой характеристики.

3.2. Архитектура реализации

Для достижения поставленной цели был разработан фаззер со следующей логикой работы (рис. 1).

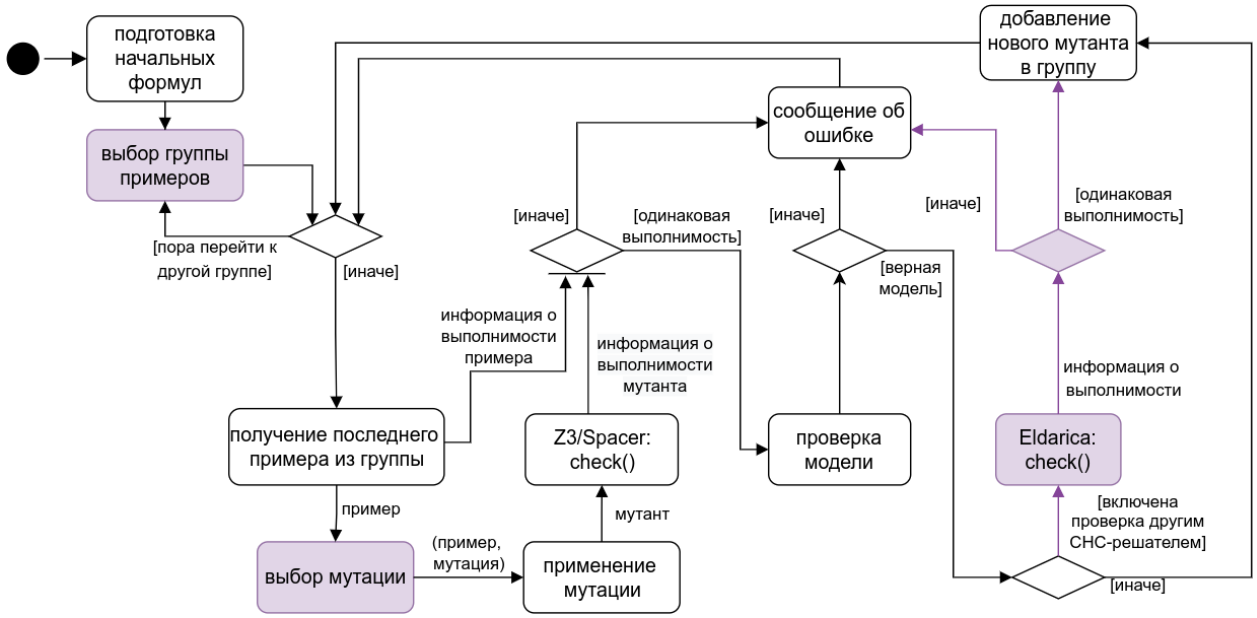

Рис. 1: Логика работы фаззера

Разработанный фаззер использует два Хорн-решателя: Spacer как основной решатель и Eldarica как решатель для кросс-валидации. Фаззер реализован на языке Python (Z3 предоставляет Python API), поскольку он дает возможность быстрой разработки. В Z3 реализована запись трассы исполнения. Сбор трассы осуществляется при сборке Z3 с флагом `-D_TRACE`. При включении сбора трассы с параметром `spacer`,

Z3 выписывает в трассу ключевые этапы решения систем дизъюнктов Хорна.

Фаззер соответствует так называемой clock-парадигме, то есть работает непрерывно, без остановки мутируя и проверяя примеры, пока не будет остановлен принудительно.

Его работа начинается с подготовки данных о начальных формулах. Каждый входной пример задаёт группу, состоящую из самого примера и его мутированных версий, то есть всё множество примеров делится на классы эквивалентности с отношением происхождения из общей начальной формулы. После подготовки начальных данных происходит выбор группы примеров. Последний член группы должен будет подвергнуться мутированию: это может быть как начальная формула, так и её последний мутант.

Далее происходит выбор мутации и её применение. Полученный мутант проверяется решателем, и полученный результат сравнивается с выполнимостью начального примера. Если результаты решения отличаются, то информация о найденной ошибке записывается в журнал. Если обе формулы выполнимы, то фаззер подставляет модель мутанта в его формулу и по очереди проверяет каждый дизъюнкт. Если какое-то из ограничений оказывается невыполнимым, то производится запись в журнал об ошибке.

Далее мутант либо проверяется Хорн-решателем Eldarica, либо добавляется в свою группу. Выбор одного из этих действий зависит от параметров запуска фаззера. В первом случае происходит сравнение результата решения Eldarica с информацией о начальной выполнимости. В случае совпадения результатов мутированная формула добавляется в группу, а в альтернативном случае сообщается о найденной ошибке.

Далее цикл повторяется, пока не потребуется сменить группу примеров. Такая потребность может возникнуть в следующих случаях.

- Трасса исполнения, порождаемая Хорн-решателем Spacer при решении мутантов из одной группы, повторилась $5 \cdot n$ раз, где n — количество дизъюнктов в системе.

- Достигнут лимит обнаруженных ошибок и случаев, когда Spacer не может решить пример или делает это слишком долго.

Цветом на схеме обозначены те блоки, которые могут не выполняться или выполняться по-разному в зависимости от опций, с которыми запущен фаззер.

- Для кросс-валидации решения начальной формулы и мутанта фаззер может использовать Хорн-решатель Eldarica.
- Выбор мутации может быть взвешенным или равновероятным.
- Выбор группы примеров, которая подвергнется изменению, зависит от заданной при запуске фаззера эвристики. Эвристика определяет, какие примеры считаются более подверженными возникновению ошибок при мутировании.

Чтобы каждый раз при запуске фаззера не тратить время на прогон входных систем дизъюнктов, информация о их выполнимости, времени решения и уникальных трассах записана. В самом начале работы фаззер считывает эти данные.

3.3. Мутации

Мутации, используемые фаззером можно разделить на три группы: предлагаемые в данной работе мутации, то есть не имеющие готовых реализаций в Z3 (мутации 1-8), переписывания Z3 (мутации 9, 10) и параметры решения (мутации 11). При запуске фаззера можно указать, какие группы мутаций он может использовать (по умолчанию фаззер использует все реализованные мутации).

1. SWAP_AND меняет местами два члена конъюнкции:

$$\phi \wedge \psi \rightsquigarrow \psi \wedge \phi.$$

2. DUP_AND дублирует один член конъюнкции:

$$\phi \wedge \psi \rightsquigarrow \phi \wedge \psi \wedge \phi.$$

3. BREAK_AND разбивает конъюнкцию на две:

$$\phi \wedge \psi \wedge \tau \rightsquigarrow \phi \wedge (\psi \wedge \tau).$$

4. SWAP_OR меняет местами два члена дизъюнкции:

$$\phi \vee \psi \rightsquigarrow \psi \vee \phi.$$

5. MIX_BOUND_VARS перемешивает переменные в кванторной приставке:

$$\forall\,(x, y, z).\psi(x, y, z) \rightsquigarrow \forall\,(y, z, x).\psi(x, y, z).$$

6. ADD_INEQ заменяет неравенство на конъюнкцию этого же и менее сильного неравенства:

$$x < c \rightsquigarrow (x < c) \wedge (x < c + 1).$$

7. ADD_LIN_RULE добавляет линейное правило, которое может быть упрощено до $\forall\,\bar{x}.\bot \rightarrow P(\bar{x})$, где P — случайно выбранный неинтерпретированный предикатный символ системы (посылка импликации — одна из собранного множества невыполнимых формул).

8. ADD_NONLIN_RULE добавляет нелинейное правило вида:

$$\forall\,\bar{v}.(\exists\,\bar{x}.\ (x_1 > x_2 \wedge P(\bar{x}, \bar{v})) \wedge (x_2 > x_3 \wedge P(\bar{x}, \bar{v})) \wedge \ldots$$
$$\wedge (x_n > x_1 \wedge P(\bar{x}, \bar{v}))) \rightarrow P(\bar{x}, \bar{v}), \text{ где}$$

   - $\bar{x} = (x_1, x_2, \ldots, x_n)$, $x_i \in \mathbb{Z}$, n — случайное число от 1 до 10;
   - $\bar{v} = (v_1, v_2, \ldots, v_m)$, причём m — арность случайно выбранного неинтерепретированного предикатного символа P, и все типы элементов $\bar{v}$ соответствуют типам принимаемых P аргументов в том же порядке.

Запись $P(\bar{x}, \bar{v})$ означает, что P принимает последовательность из m случайных аргументов объединения $\bar{x}$ и $\bar{v}$ в соответствии с допустимыми типами аргументов (хотя бы одна такая последовательность точно существует — это $\bar{v}$).

9. Эквивалентные преобразования, которые предлагает API Z3. Используются следующие параметры этих преобразований:

   - arith_ineq_lhs;
   - arith_lhs;
   - blast_distinct;
   - blast_select_store;
   - elim_and;
   - elim_rem;
   - elim_to_real;
   - eq2ineq;
   - expand_power;
   - expand_select_ite;
   - expand_select_store;
   - expand_store_eq;
   - expand_tan;
   - gcd_rounding;
   - hoist_ite;
   - hoist_mul;
   - ite_extra_rules;
   - local_ctx;
   - mul2concat;
   - mul_to_power;
   - pull_cheap_ite;
   - push_ite_arith;
   - rewrite_patterns;
   - som;
   - sort_store;
   - sort_sums;
   - split_concat_eq;
   - algebraic_number_evaluator;
   - elim_ite;
   - elim_sign_ext;
   - flat;
   - push_to_real;
   - ignore_patterns_on_ground_qbody.

10. Функция simplify() без параметров (эквивалентное переписывание, в основном используемое во внутренней структуре Z3).

11. Изменение параметров решателя, влияющих на ход решения примера:

- spacer.ctp;
- spacer.elim_aux;
- spacer.eq_prop;
- spacer.ground_pobs;
- spacer.keep_proxy;
- spacer.mbqi;
- spacer.propagate;
- spacer.reach_dnf;
- spacer.use_array_eq_generalizer;
- spacer.use_derivations;
- spacer.use_inc_clause;
- spacer.use_inductive_generalizer;
- xform.coi;
- xform.compress_unbound;
- xform.inline_eager;
- xform.inline_linear;
- xform.slice;
- xform.tail_simplifier_pve.
- spacer.p3.share_invariants;
- spacer.p3.share_lemmas;
- spacer.use_lim_num_gen;
- spacer.reset_pob_queue;
- spacer.simplify_lemmas_post;
- spacer.simplify_lemmas_pre;
- spacer.simplify_pob;
- spacer.use_bg_invs;
- spacer.use_euf_gen;
- spacer.use_lemma_as_cti;
- xform.array_blast_full;
- xform.coalesce_rules;
- xform.elim_term_ite;
- xform.inline_linear_branch;
- xform.instantiate_arrays;
- xform.instantiate_arrays.enforce;
- xform.instantiate_quantifiers;
- xform.quantify_arrays;
- xform.transform_arrays.

Если фаззер запущен с опцией взвешенного выбора мутаций, то веса мутаций обновляются в течение всей работы фаззера в зависимости от того, приводит ли мутация к открытию новой трассы исполнения при решении примера.

3.4. Сортировка примеров

Обнаружение ошибок мутационными фаззерами во многом определяется выбором начальных входных данных [25]. В разработанном фаззере реализовано несколько способов сортировки примеров, основанных на разных идеях о том, мутация какого примера вероятнее приведет к обнаружению ошибки. При запуске фаззера можно задать любую из следующих четырёх эвристик, а так же совмещать их, если есть потребность, чтобы по одной эвристике примеры были разбиты по группам, а по другой упорядочены в каждой группе.

1. Предпочтение примеров, покрывающих при решении наиболее редкие переходы.

2. Выбор примеров по редким состояниям.

3. Выбор наиболее сложных примеров. По этой эвристике нелинейные системы предпочитаются линейным, а внутри этих групп они упорядочиваются по количеству неинтерпретированных предикатных символов.

4. Выбор наиболее простых примеров.

Остановимся подробнее на первых двух пунктах. Решатель может быть представлен как система, которая изменяет свое состояние в дискретные моменты времени. Состояния в этом контексте — этапы решения, отраженные в трассе исполнения (раздел 2.5), а моменты времени — строчки трассы. Собрав статистику решения входных формул, можно узнать, какие состояния и какие переходы (то есть пары состояний) встречаются реже других.

Пусть каждое состояние решателя имеет номер, n — количество состояний. В случае с выбором примеров, при решении которых совершаются редкие переходы, составляется матрица перехода P, то есть такая стохастическая матрица, на пересечении i-ой строки и j-го столбца ко-

торой находится вероятность перехода из i-го состояния в j-ое — p_{ij}:

$$P = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{pmatrix}$$

По этой матрице может быть получена матрица весов W, где вес перехода обратно пропорционален вероятности перехода (если вероятность перехода равна 0, то вес тоже приравнивается к 0):

$$w_{ij} = \begin{cases} \frac{1}{p_{ij}}, \text{ если } p_{ij} > 0 \\ 0 \quad \text{иначе} \end{cases}$$

Поэлементно перемножив матрицу весов с матрицей T_m, отражающей количество переходов конкретного примера (m — его номер), и взяв сумму всех элементов результата, мы получим значение, означающее приоритет этого примера:

$$k_m = \sum_{1 \leqslant i,j \leqslant n} w_{ij} \cdot t_{ij}^m$$

В случае с выбором примеров по редким состояниям совершаются аналогичные действия с векторами.

3.5. Сокращение цепочки мутаций и примера

После обнаружения ошибок в решателе важно сообщить о найденных проблемах разработчикам. Так как человек лучше воспринимает и анализирует более простые случаи возникновения ошибок, то появляется необходимость в сокращении проблемного примера и цепочки мутаций, применение которых приводит к ошибке (рис. 2).

Для этих целей был создан инструмент, который является реализацией обоих способов упрощения условий возникновения ошибки.

Сокращение цепочки мутаций в этом инструменте осуществляется по алгоритму дельта-отладки [24]. Он состоит из следующих действий.

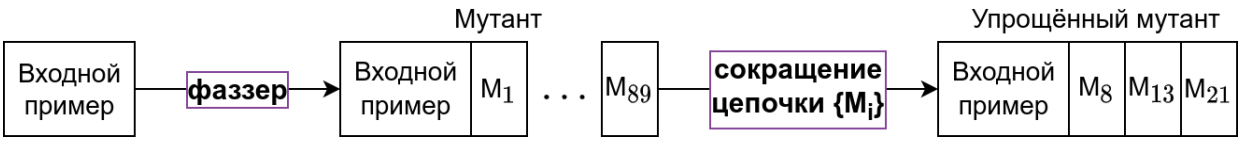

Рис. 2: Схема сокращения цепочки мутаций.

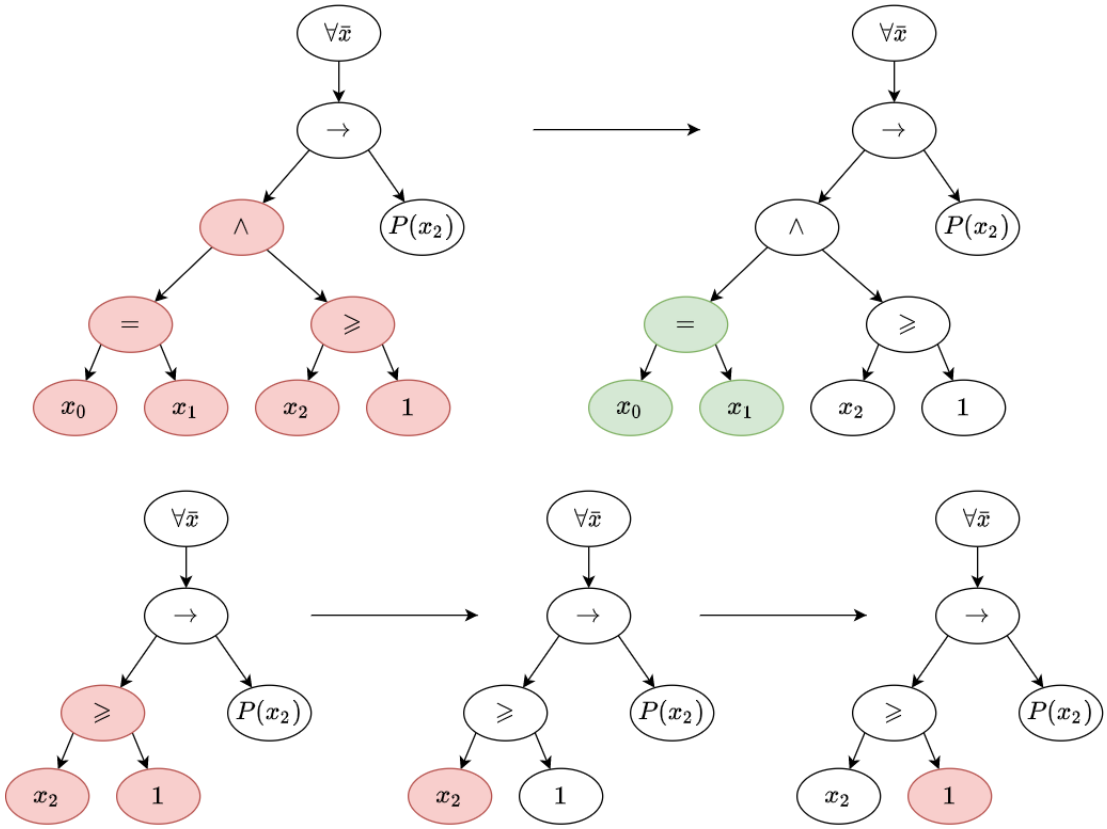

Рис. 3: Схема сокращения формулы, на которой возникает ошибка.

1. Цепочка мутаций делится на две равные части.

2. Каждая часть рассматривается в отдельности, и если без неё ошибка всё равно появляется, то эта часть цепочки может быть удалена.

3. Оставшиеся части цепочки делятся пополам, предыдущий и этот шаг повторяются.

Сокращение проблемного примера производится посредством удаления подвыражений из уменьшаемого мутанта по алгоритму иерархической дельта-отладки [17]. Этот алгоритм отличается от обычной дельта-отладки тем, что вместо исключения подпоследовательностей

происходит удаление поддеревьев выражения. То есть сначала предпринимается попытка удалить дизъюнкт Хорна из системы целиком. Если это сделать нельзя, то инструмент пробует удалить его потомков и так далее. При исключении частей абстрактного синтаксического дерева примера необходимо следить за тем, чтобы мутант оставался эквивалентен своей начальной формуле и чтобы ошибка оставалась воспроизводимой.

На рисунке 3 представлен пример такого упрощения. Посылка импликации не может быть удалена, так как импликация — бинарная операция. Однако с сохранением эквивалентности и возникновения ошибки может быть удален её потомок $x_0 = x_1$. При сокращении $x_2 \geqslant 1$ либо нарушается эквивалентность формулы начальной, либо перестаёт возникать ошибка. Потомки неравенства не могут быть удалены, так как «$\geqslant$» — бинарное отношение.

4. Экспериментальные данные

В этом разделе приведены данные о найденных в решателе ошибках и оценивается эффективность фаззера.

Замеры проводились на компьютере с Arch Linux x86_64, Intel Core i7-4790 CPU, 3.60GHz, 32Gb RAM.

4.1. Входные данные

Начальные примеры, подвергающиеся мутациям, были взяты из следующих источников:

- набор бенчмарков с соревнования решателей систем дизъюнктов Хорна с ограничениями CHC-COMP за 2021-ый год [26];
- набор бенчмарков с международного конкурса по верификации программного обеспечения SV-COMP [27];
- набор бенчмарков для статей [18, 6].

Все входные примеры имеют SMT-LIB2 формат [2]. Среди них были выбраны те бенчмарки, которые Spacer может решить и решение которых укладывается в 2 секунды. Получившийся набор бенчмарков состоит из 3404 систем дизъюнктов Хорна с ограничениями в теориях линейной вещественной и целочисленной арифметики и теории массивов.

4.2. Обнаруженные ошибки

Разработанный фаззер обнаружил ошибки в решателе Spacer: 11 из них уже устранены разработчиками Z3, а другие находятся в процессе исправления.

Следующие ошибки исправлены.

- При удвоении одного элемента конъюнкции (мутация DUP_AND)

результат решения системы менялся с «выполнимо» на «невыполнимо»[3].

- При решении примера с параметром решателя fp.xform.array_blast результат менялся с «выполнимо» на «невыполнимо»[4]. При решении системы с указанным параметром, в примерах в теории массивов пары равенств вида $(v_1 = select\ A\ i_1) \wedge (v_2 = select\ A\ i_2)$ заменялись на $(i_1 = i_2 \rightarrow v_1 = v_2)$ (редукция Аккермана [8]), где, A — массив, параметризованный множеством индексов I и множеством значений V, $i_1, i_2 \in I$, $v_1, v_2 \in V$.

- В некоторых случаях решатель строил неправильную модель[5].

Группа случаев неправильного построения модели связана с ошибками в трансформациях дизъюнктов. Не все найденные ошибки такого вида устранены[6], так как их исправление требует существенных и продуманных изменений в решателе.

Чтобы исправленные ошибки построения модели не возникали вновь, примеры, вызвавшие эти ошибки, были добавлены в систему тестирования Z3 в качестве регрессионных тестов. Для этого в систему тестирования Z3 была встроена проверка модели на этих примерах[7].

4.3. Оценка качества фаззера

При оценке работы фаззера важным показателем является не только количество найденных ошибок, но и статистика открытия уникальных трасс исполнения. С помощью этой метрики можно отследить, ис-

[3] https://github.com/Z3Prover/z3/issues/5714 [accessed: 6 мая 2022 г.]
[4] https://github.com/Z3Prover/z3/issues/5833 [accessed: 6 мая 2022 г.]
[5] https://github.com/Z3Prover/z3/issues/5858 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5862 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5863 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5865 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5866 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5869 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5874 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5882 [accessed: 6 мая 2022 г.]
https://github.com/Z3Prover/z3/issues/5903 [accessed: 6 мая 2022 г.]
[6] https://github.com/Z3Prover/z3/issues/5920 [accessed: 6 мая 2022 г.]
[7] https://github.com/Z3Prover/z3test/pull/46 [accessed: 6 мая 2022 г.]

следует ли фаззер разнообразные примеры, или ход решения генерируемых им формул почти всегда совпадает, что фактически означает, что фаззер не анализирует разные варианты поведения решателя.

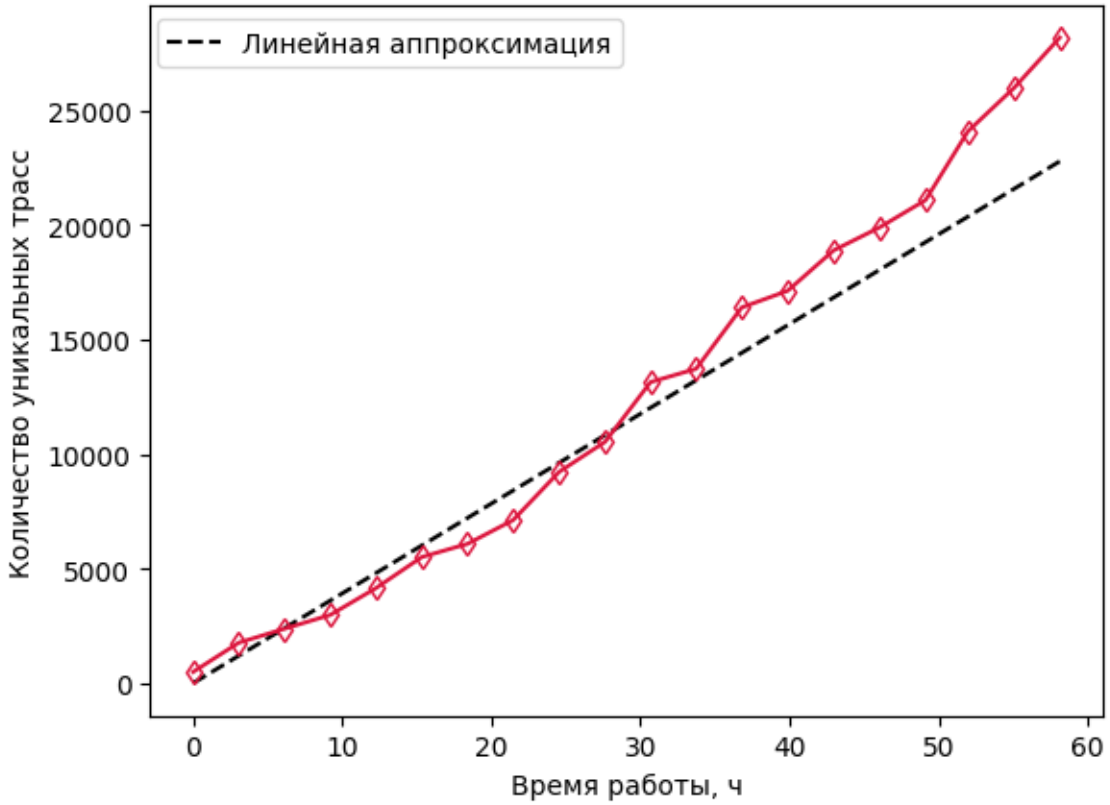

Рис. 4: Результат прогона фаззера с использованием всех мутаций, взвешенным выбором мутаций и приоритизацией примеров по редким переходам.

Зависимость числа открытых трасс исполнения от времени работы фаззера представлена на графике 4. По нему видно, что эта зависимость практически линейна, и частота открытия новых трасс исполнения не снижается по мере работы фаззера. Это означает, что фаззер действительно исследует поведение решателя на разных сценариях решения систем дизъюнктов.

Заключение

В рамках данной работы были получены следующие результаты.

- Предложен способ фаззинга Хорн-решателя на основе методики метаморфического тестирования.
- Спроектирован и разработан фаззер[8] для тестирования решателя Spacer на языке Python с четырьмя эвристиками сортировки примеров, возможностью взвешенного выбора мутаций, а также использованием решателя систем дизъюнктов Хорна Eldarica для кросс-валидации.
- С помощью спроектированного фаззера обнаружено 15 ошибок в Хорн-решателе Spacer. Все ошибки подтверждены разработчиками Z3: 11 из них устранены, а другие находятся в процессе исправления. Исправленные ошибки были добавлены в систему регрессионного тестирования Z3.
- Реализовано сокращение проблемных примеров и цепочек мутаций.

[8] https://github.com/AnzhelaSukhanova/spacer_fuzzer [accessed: 6 мая 2022 г.]

Список литературы

[1] Scott Joseph, Sudula Trishal, Rehman Hammad, Mora Federico, and Ganesh Vijay. BanditFuzz: Fuzzing SMT Solvers with Multi-agent Reinforcement Learning // Formal Methods / ed. by Huisman Marieke, Păsăreanu Corina, and Zhan Naijun. — Cham : Springer International Publishing. — 2021. — P. 103–121.

[2] Barrett Clark, Stump Aaron, and Tinelli Cesare. The SMT-LIB Standard: Version 2.0 // Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK) / ed. by Gupta A. and Kroening D. — 2010.

[3] Brummayer Robert and Biere Armin. Fuzzing and delta-debugging SMT solvers // ACM International Conference Proceeding Series. — 2009. — 01. — P. 1–5.

[4] Böhme Marcel, Pham Van-Thuan, and Roychoudhury Abhik. Coverage-Based Greybox Fuzzing as Markov Chain // IEEE Transactions on Software Engineering. — 2019. — Vol. 45, no. 5. — P. 489–506.

[5] The CHC competition (CHC-COMP) [Электронный ресурс]. — Access mode: https://chc-comp.github.io/ (online; accessed: 30.04.2022).

[6] Champion Adrien, Kobayashi Naoki, and Sato Ryosuke. HoIce: An ICE-Based Non-linear Horn Clause Solver: 16th Asian Symposium, APLAS 2018, Wellington, New Zealand, December 2–6, 2018, Proceedings. — 2018. — 01. — P. 146–156. — ISBN: 978-3-030-02767-4.

[7] Chen T. Y., Cheung S. C., and Yiu S. M. Metamorphic Testing: A New Approach for Generating Next Test Cases. — 2020. — Access mode: https://arxiv.org/abs/2002.12543.

[8] Komuravelli Anvesh, Bjorner Nikolaj, Gurfinkel Arie, and McMillan Kenneth L. Compositional Verification of Procedural Programs us-

ing Horn Clauses over Integers and Arrays. — 2015. — Access mode: https://arxiv.org/abs/1508.01288.

[9] Mansur Muhammad Numair, Christakis Maria, Wüstholz Valentin, and Zhang Fuyuan. Detecting Critical Bugs in SMT Solvers Using Blackbox Mutational Fuzzing // Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. — New York, NY, USA : Association for Computing Machinery. — 2020. — ESEC/FSE 2020. — P. 701–712. — Access mode: https://doi.org/10.1145/3368089.3409763.

[10] Fedyukovich Grigory and Rümmer Philipp. Competition Report: CHC-COMP-21 // Electronic Proceedings in Theoretical Computer Science. — 2021. — Sep. — Vol. 344. — P. 91–108. — Access mode: http://dx.doi.org/10.4204/EPTCS.344.7.

[11] Pacheco Carlos, Lahiri Shuvendu K., Ernst Michael D., and Ball Thomas. Feedback-Directed Random Test Generation // 29th International Conference on Software Engineering (ICSE'07). — 2007. — P. 75–84.

[12] Yao Peisen, Huang Heqing, Tang Wensheng, Shi Qingkai, Wu Rongxin, and Zhang Charles. Fuzzing SMT Solvers via Two-Dimensional Input Space Exploration // Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. — New York, NY, USA : Association for Computing Machinery. — 2021. — ISSTA 2021. — P. 322–335. — Access mode: https://doi.org/10.1145/3460319.3464803.

[13] Gurfinkel Arie and Bjørner Nikolaj. The Science, Art, and Magic of Constrained Horn Clauses // 2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). — 2019. — P. 6–10.

[14] Hojjat Hossein and Rümmer Philipp. The ELDARICA Horn Solver // 2018 Formal Methods in Computer Aided Design (FMCAD). — 2018. — P. 1–7.

[15] Bjørner Nikolaj, Gurfinkel Arie, McMillan Ken, and Rybalchenko Andrey. Horn Clause Solvers for Program Verification // Fields of Logic and Computation II: Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday / ed. by Beklemishev Lev D., Blass Andreas, Dershowitz Nachum, Finkbeiner Bernd, and Schulte Wolfram. — Cham : Springer International Publishing, 2015. — P. 24–51. — ISBN: 978-3-319-23534-9. — Access mode: https://doi.org/10.1007/978-3-319-23534-9_2.

[16] Komuravelli Anvesh, Gurfinkel Arie, and Chaki Sagar. SMT-based Model Checking for Recursive Programs. — 2014. — 1405.4028.

[17] Misherghi Ghassan and Su Zhendong. HDD: Hierarchical Delta Debugging // Proceedings of the 28th International Conference on Software Engineering. — New York, NY, USA : Association for Computing Machinery. — 2006. — ICSE '06. — P. 142–151. — Access mode: https://doi.org/10.1145/1134285.1134307.

[18] Mordvinov Dmitry and Fedyukovich Grigory. Verifying Safety of Functional Programs with Rosette/Unbound. — 2017. — Access mode: https://arxiv.org/abs/1704.04558.

[19] Mordvinov Dmitry and Fedyukovich Grigory. Property Directed Inference of Relational Invariants // 2019 Formal Methods in Computer Aided Design (FMCAD). — 2019. — P. 152–160.

[20] Offutt Jeff and Xu Wuzhi. Generating Test Cases for Web Services Using Data Perturbation // SIGSOFT Softw. Eng. Notes. — 2004. — sep. — Vol. 29, no. 5. — P. 1–10. — Access mode: https://doi.org/10.1145/1022494.1022529.

[21] Rümmer Philipp. A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic. — 2008. — 11. — Vol. 5330. — P. 274–289.

[22] Satake Yuki, Unno Hiroshi, and Yanagi Hinata. Probabilistic Inference for Predicate Constraint Satisfaction // Proceedings of the AAAI Conference on Artificial Intelligence. — 2020. — 04. — Vol. 34. — P. 1644–1651.

[23] Sutton Michael, Greene Adam, and Amini Pedram. Fuzzing: Brute Force Vulnerability Discovery. — 2007. — 01.

[24] Donaldson Alastair F., Thomson Paul, Teliman Vasyl, Milizia Stefano, Maselco André Perez, and Karpiński Antoni. Test-Case Reduction and Deduplication Almost for Free with Transformation-Based Compiler Testing // Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. — New York, NY, USA : Association for Computing Machinery. — 2021. — PLDI 2021. — P. 1017–1032. — Access mode: https://doi.org/10.1145/3453483.3454092.

[25] Yoo S. and Harman M. Regression Testing Minimization, Selection and Prioritization: A Survey // Softw. Test. Verif. Reliab. — 2012. — mar. — Vol. 22, no. 2. — P. 67–120. — Access mode: https://doi.org/10.1002/stv.430.

[26] The benchmarks selected at CHC-COMP 2021 [Электронный ресурс]. — Access mode: https://github.com/chc-comp/chc-comp21-benchmarks (online; accessed: 30.04.2022).

[27] The benchmarks selected at last SV-COMP [Электронный ресурс]. — Access mode: https://gitlab.com/sosy-lab/benchmarking/sv-benchmarks/-/tree/main/clausess (online; accessed: 30.04.2022).

[28] de Moura Leonardo and Bjørner Nikolaj. Z3: An Efficient SMT Solver // Tools and Algorithms for the Construction and Analysis of

Systems / ed. by Ramakrishnan C. R. and Rehof Jakob. — Berlin, Heidelberg : Springer Berlin Heidelberg. — 2008. — P. 337–340.