Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.М07-мм

Двунаправленное направляемое конфликтами резюмирование кода в символьной виртуальной машине V#

Седлярский Михаил Андреевич

Отчёт по проекто-технологической практике

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н., Д. А. Мордвинов

Санкт-Петербург
2022

Оглавление

Введение

С каждым днём программное обеспечение становится всё сложнее и сложнее. Тестов, написанных людьми, уже недостаточно, чтобы исчерпывающе проверять программы на корректность. Пропущенные программные ошибки в различных сферах человеческой деятельности могут приводить к большим потерям: начиная от задержек во времени и заканчивая человеческими жизнями. В 2002 году Грегори Тассей оценивал годовой ущерб экономике США от ошибок программного обеспечения в 59.5 миллиардов долларов США [11].

Среди формальных методов можно выделить символьное исполнение. Это метод моделирования работы программы с использованием символьных значений вместо конкретных. Во время символьного исполнени, встретив оператор ветвления, анализатор добавит условие (ограничение) к себе в список и продолжит анализ в обеих ветвях. Затем используются SMT решатели, которые по заданным условиям находят входные данные, приводящие программу в интересующие ветви. Такими ветвями могут быть, например, те, в которых выбрасываются исключения.

Одной из главных проблем символьного исполнения является, так называемый, комбинаторный взрыв путей. Количество путей программы растёт экспоненциально от её размера. Более того, если программа содержит циклы с заранее неизвестным количеством итераций или рекурсию, то количество возожных путей исполнения будет бесконечным.

Один из способов борьбы с обозначенной проблемой заключается в резюмировании (summarization) участков кода. Если некоторый блок программы исполняется многократно, то символьная виртуальная машина может его выполнить единожды, а в последующие разы использовать краткую сводку, описывающую целое множество состояний программы. Такими блоками обычно являются тела циклов и функции.

Резюмирование позволяет анализировать большую кодовую базу с помощью композиции сводок небольших фрагментов исходной программы. Таким образом, отпадает необходимость в символьном исполнении

всего кода разом.

Целью работы является реализация резюмрования кода на основе двунаправленного поиска в символьной виртуальной машине для платформы .net core.

1. Постановка задачи

Целью работы является разработка и реализация алгоритма двунаправленного направляемого конфликтами резюмирования кода в символьной виртуальной машине для платформы .net core. Для выполнения цели были поставлены следующие задачи:

1. реализовать извлечение unsat-ядер из ограничений пути;
2. реализовать двунаправленные эвристики на основе unsat-ядер;
3. реализовать двунаправленное символьное исполнение в виртуальной машине V#;

2. Обзор

Чтобы рассмотреть существующие подходы в резюмировании, стоит выделить критерии сравнения. Алгоритмы могут различаться по степени аппроксимации, способам представления и обобщения.

2.1. Степень аппроксимации

По степени аппроксимации подходы в резюмировании делятся на два вида: слабо- и сверх- аппроксимирующие. Слабо аппроксимирующие резюме описывают подмножество возможных состояний. Состояния, включённые в подмножество, достижимы. Но слабо аппроксимирующее обобщение часто охватывает не все достижимые состояния. В противовес, сверх-аппроксимирующие обобщения включают всевозможные достижимые состояния и, как правило, ряд недостижимых. Такие резюме удобны для доказательства недостижимости опредёлённых точек в программах. Одни из первых работ, посвящённых резюмированию кода в символьном исполнении, использовали слабо апроксимирующие резюме [1, 6]. В большинстве последующих работ также использовались слабые аппроксимации, чтобы повысить производительность символьного выполнения [10, 2, 5]. Сверх-аппроксимирующие обобщения зачастую используют в области верификации программ.

Для ясности рассмотрим простую функцию модуля и её аппроксимации: слабую 1 и сверхсильную 2

Листинг 1: Функция abs

```
int abs(int x) {
    return x < 0 ? -x : x;
}
```

$$x < 0 \wedge abs(x) = -5 \tag{1}$$

$$x \neq INT\_MIN \Longrightarrow abs(x) > 0 \tag{2}$$

Стоит отметить, что в первом случае обобщение исключает множество состояний с положительной переменной. Например, $x = 3$. Во втором случае, напротив, помимо верных состояний мы включаем и те, что никода не смогут быть достигнуты. Например, выражению 2 удовлетворяет $x = -123$.

В работе будут использоваться сверх-аппроксимирующее резюмирование т.к. с его помощью можно более эффективно обрезать пространстов возможных состояний.

2.2. Способ представления

Резюме могут обобщать бесконечное число состояний, поэтому требуется достаточно компактный и универсальный способ их представления. Одним из самых распространённых способов является запись обобщений в виде формул логики первого порядка. Для записи сверх-аппроксимирующих резюме лучше всего подходят формулы в КНФ т.к. современные SMT решатели очень эффективно обрабатывают формулы в конъюнктивной форме. Более того, КНФ для сверх-аппроксимирующих обобщений будет занимать меньше места по сравнению с ДНФ. Существуют другие способы представления резюме, отличные от формул логики первого порядка [5, 3, 9]. Но т.к. область их применения довольно ограничена, в данной работе будут использоваться обобщения на основе формул.

2.3. Способ обобщения

От способа обобщения формулы зависит качество резюме.

Одним из распространённых способов обобщения является интерполяция Крейга [4], которая позволяет обобщать формулы за линейное время. Но современные SMT решатели довольно медленно решают задачу интерполяции битовых векторов. Поэтому в данной работе используется альтернативный подход, основанный на unsat ядрах [7].

3. Метод

3.1. Двунаправленное исполнение

Ранее в символьной виртуальной машине V# уже был реализован механизм двунаправленного исполнения, но со временем его концепция была несколько пересмотрена. В связи с этим появилась необходимость реинжиниринга кодовой базы. В этом разделе представлено краткое изложение сути двунаправленного исполнения.

На вход алгоритма подаётся программа в виде байт-кода и набор целевых инструкций. Символьная виртуальная машина будет искать пути до обозначенных целей. Для максимизации покрытия достаточно передать в качестве целей все инструкции программы.

Двунаправленное исполнение чередует прямые и обратные шаги.

Стратегия прямого исполнения заключается в том, что бы найти путь от начальной точки программы в обозначенные цели. Если прямое исполнение за некоторое число попыток не может прийти к тем или иным целям, то тогда символьная машина переключается на стратегию обратного шага.

Стратегия обратного исполнения заключается в том, что бы построить ограничения пути не от начальной точки программы к цели, а наоборот. Для этого стратегия обратного поиска находит точку обратного распространения (init point). Затем от этой точки до целей с чистого листа запускается прямое исполнение. Т.е. исполнение происходит изолировно от прдыдущих значений в символьной памяти. После достижения целей, мы получаем ограничения пути, называемые наислабейшим предусловием. Затем, обратное исполнение распространяет наислабейшее предусловие в предшествующй контекст и перемещает достигнутые цели в точку обратного распространения. Таким образом, вместо приближения прямого исполнения к целям, мы приблизили цели к точке входа в программу. Важно отметить, что обратное исполнение не интерпретирует инструкции байт-кода в обратном порядке.

Чередование прямых и обратных шагов позволяет символьной вир-

туальной машине быстрее достигать назначенных целей.

Двунаправленные стратегии должны эффективно находить точки обратного распространения, чтобы «протаскивать» цели к входу в программу. В последующих разделах предлагается стратегия поиска точек, основанная на конфликтах в ограничении пути.

3.2. UNSAT ядра

Запишем конъюнкцию формул $c_1, ..., c_n$ как множество $PC = \{c_1, ..., c_n\}$. Пусть PC невыполнимо в некой теории Г. Тогда минимальное невыполнимое (unsat) ядро формулы PC это – минимальное невыполнимое в теории Г подмножество PC. Извлечение невыполнимых ядер реализовано во многих современных SMT решателях. Например, в Z3 [12].

3.3. Эвристики на основе unsat-ядер

Качество двунаправленного исполнения напрямую зависит от стратегии извлечения точек обратного распространения.

Когда прямая стратегия исполнения не может прийти к целям, как правило, у множества путей программы наблюдаются невыполнимые ограничения. Пути натыкаются на конфликты в многочисленных условиях в коде и не могут продвинуться дальше.

Одной из гипотез данной работы является то, что лучше всего на роль точек обратного распространения подходят атомы unsat ядер ограничений пути.

Было рассмотрено две эвристики использования unsat ядер.

Target-based эвристика. Суть подхода в том, что при каждом возникающем unsat конфликте извлекать из ядра случайный атом и сапоставлять ему текущие цели, к которым стремился путь. Если целей нет, то новой точки обратного распространения не создаётся. В следующем разделе будет показана неэффективность такой эвристики.

Call-chain эвристика. Этот подход отличается от предыдущего тем, что создаёт точки распространения независимо от наличия целей у застопорившегося пути. Каждый раз, когда возникает unsat, из его

ядра берётся два атома: первый и последний. Порядок атомов определяется хронологией их добавления в ограничения пути. У каждого атома хранится история вызовов функций, которая позволяет примерно понять путь, по которому шло символьное исполнение. В истории двух атомов находится общий метод. Затем выстраивается цепочка точек обратного распространения,

$$(atom_{first}, p_1), (p_1, p_2), ..., (p_n, atom_{last}) \tag{3}$$

где цель каждой точки это начало предыдущей. Таким образом, стратегия прямого поиска сможет в изоляции дойти от одного атома до другого. Затем, от последнего атома по тем же принципам выстраиваются цепочки к изначальным целям пути, если такие были. Преимущество эвристики в том, что создаётся больше точек обратного распространения и они чаще попадают в начала блоков кода: циклов или функций.

4. Эксперимент

Проверим степень утилизации unsat ядер в target-based подходе.

4.1. Условия эксперимента

Символьнпая машина будет запускаться на 511 неболших тестовых программах, написанных на языке C#. Данные тесты моделируют варьюируются по сложности от abs до regExp.

4.2. Исследовательские вопросы

Узнать насколько эффективно используются unsat ядра в обозначенном подходе

4.3. Метрики

Хорошей утилизацией unsat ядер будет считаться использование свыше 70% из их общего числа.

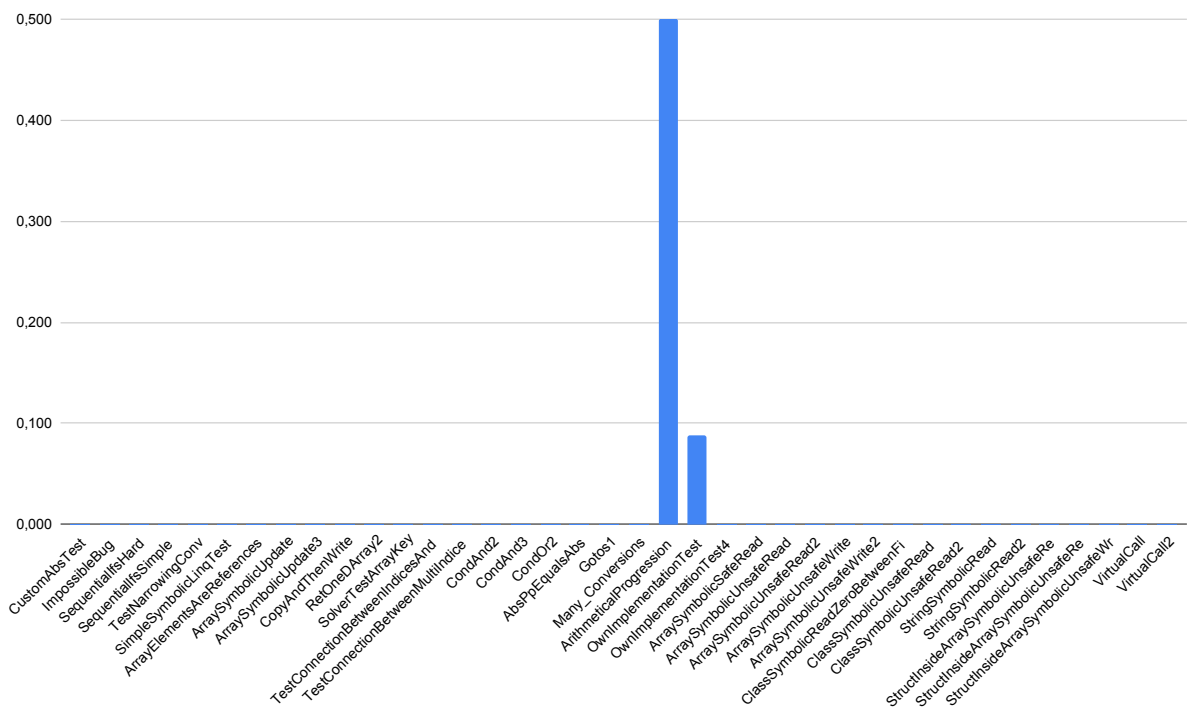
4.4. Результаты

Из 511 тестов только в 38 появляются невыполнимые ограничения пути. На графике 1 избражена доля использованных unsat ядер. Как видно из графика, только 2 из 38 тестов смотрят на информацию, извлекаюемую из конфликтов. В среднем, в target-based подходе отбрасывается $98,5\%$ ядер.

4.5. Обсуждение результатов

Данный результат указывает на довольно неэффективное использование полученных данных в target-based подходе в то время, как в call-chain подходе используются все unsat ядра.

Рис. 1: График утилизации unsat ядер в target-based эвристике. По оси X тесты, по оси Y доля использованных unsat ядер от общего числа



5. Применение

Был написан каркас двунаправленного символьного исполнения, опробована одна из эвристик обнаружения точек обратного распространения. Ещё одна стратегия сформулирована и требует реализации. Данные результаты являются фундаментом для последующего механизма резюмирования кода в символьном исполнении.

6. Реализация

В этом разделе представлено описание проделанной работы в кодовой базе символьной виртуальной машины V#. Все азименения в коде, внесённые автором, можно посмотреть по ссылке на GitHub [8].

Извлечение unsat ядер. В чистом виде unsat ядро, полученное из SMT решателя, малоинформативно. Оно является массивом предикатов. Был переписан модуль PathConstraint, чтобы можно было локализовать атомы ядра точками в коде. Каждое ограничение пути теперь обладает полями codeLocation – местом в программе, где встретилось ограничение и createdAt – порядковым числом, которое у последующих ограничений пути будет строго больше. Подобные логические часы позволяют упорядычивать атомы unsat ядер.

В процессе работы было также выявлено несколько ошибок в кодировании формулы ограничений пути.

Двунаправленное исполнение и эвристики. Были расширены и видоизменены интерфейсы прямой и обратной стратегий исполнения. Был добавлен интерфейс InitPointExtractor – стратегия извлечения точек обратного рапространения две реализации данного интерфейса. Было переписано исполнение состояний в изоляции. Текущая архитектура двунаправленного испонления позволяет комбинировать разные стратегии прямого, обратного исполнения вместе с разными стратегиями извлечения точек распространения. Такой подход раскрывает возможности для различных экспериментов. Также была найдена ошибка, связанная с утечкой ссылки. Для её исправления был добавлен метод глубокого копирования объектов типа cilState.

Прочее. Также было внедрена мемоизация сущностей типа codeLocation. Т.к. это одни из самых часто создаваемых объектов, разумно не создавать многократно один и тот же codeLocation. Для удобства отладки было добавлено логирование всех шагов двунаправленного исполнения с подробным выводом сопутствующей информации.

Заключение

В осеннем семестре были выполнены следующие задачи.

- Рассмотрены актуальные публикации по теме резюмирования кода в символьном исполнении.
- Был проведён поиск и анализ shim-изоляторов для платформы .net core
- Реализована загузка зависимостей asp.net core приложения при исполнении виртуальной машиной V#.

В ходе данного семестра были выполнены следующие задачи.

- Реализовано извлечение unsat ядер из ограничений пути.
- Написан каркас двунаправленного исполнения.
- Реализовано несколько эвристик излвечения точек обратного распространения на основе unsat ядер.

В следующем семестре планируется продолжить поиски эффективной эвристики нахождения точек обраного распространения, реализовать базу данных лемм, реализовать механизм распространения относительно индуктивных лемм, апробировать полуенное решение.

Список литературы

[1] Anand Saswat, Godefroid Patrice, Tillmann Nikolai. Demand-Driven Compositional Symbolic Execution // TACAS. — 2008.

[2] Boonstoppel Peter, Cadar Cristian, Engler Dawson R. RWset: Attacking Path Explosion in Constraint-Based Test Generation // TACAS. — 2008.

[3] Compositional shape analysis by means of bi-abduction / Cristiano Calcagno, Dino Distefano, Peter W. O'Hearn, Hongseok Yang // POPL '09. — 2009.

[4] Craig William Lane. Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem // J. Symb. Log. — 1957. — Vol. 22. — P. 250–268.

[5] Dergachev Andrey, Sidorin Anton V. Summary-based method of implementing arbitrary context-sensitive checks for source-based analysis via symbolic execution. — 2016.

[6] Godefroid Patrice. Compositional dynamic test generation // POPL '07. — 2007.

[7] Guthmann Ofer, Strichman Ofer, Trostanetski Anna. Minimal unsatisfiable core extraction for SMT // 2016 Formal Methods in Computer-Aided Design (FMCAD). — 2016. — P. 57–64.

[8] MihailITPlace/VSharp. — https://github.com/VSharp-team/VSharp/compare/master...MihailITPlace:bidirectional-searcher?expand=1.

[9] Predator: A Tool for Verification of Low-Level List Manipulation - (Competition Contribution) / Kamil Dudka, Petr Müller, Petr Peringer, Tomás Vojnar // TACAS. — 2013.

[10] A Survey of Symbolic Execution Techniques / Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia et al. // ACM Computing Surveys (CSUR). — 2018. — Vol. 51. — P. 1 – 39.

[11] The Economic Impacts of Inadequate Infrastructure for Software Testing : Planning Report 02-3 / National Institute of Stan- dards and Technology, ; Executor: Gregory Tassey : 2002.

[12] de Moura Leonardo Mendonça, Bjørner Nikolaj. Z3: An Efficient SMT Solver // TACAS. — 2008.