

Санкт–Петербургский государственный университет
Кафедра системного программирования
Группа 21.Б15-мм

Бучин Вячеслав Андреевич

Взаимодействие решателя для типов и виртуальной символьной машины

Отчёт по производственной практике
в форме «Решение»

Научный руководитель:
кандидат физико-математических наук
Д.А. Мордвинов
Консультант:
инженер-исследователь кафедры системного программирования
М.П. Костицын

Санкт-Петербург
2024 г.

Содержание

Введение	3
Постановка задачи	5
1. Обзор	6
1.1. Символьное исполнение	6
1.2. SMT-решатели	8
1.3. Проект V#	9
Список литературы	11

Введение

Каждый год увеличивается объем используемого программного обеспечения. Важным этапом цикла его разработки является контроль качества. Одна из его составляющих — модульное тестирование. Для автоматизации данного процесса существует множество различных техник и подходов. Одним из которых является символьное исполнение.

Символьное исполнение [1, 5, 6] — техника, которая позволяет свести задачу анализа программного кода в условиях неопределенности входных данных к задаче проверки выполнимости формул логики первого порядка. Эти формулы представляют из себя ограничения, выполнение которых однозначно задает ветвь исполнения программы. Они могут быть устроены нетривиальным образом, а именно содержать конъюнкции, отрицания, а также дизъюнкции. Кроме того, среди ограничений можно выделить те, для решения которых необходимо иметь информацию о системе типов языка исследуемой программы — *ограничения на типы*.

Для их решения используются различные решатели: для ограничений, касающихся типов используются решатели для типов, а для оставшихся — SMT-решатели (Satisfiability Modulo Theories) [2, 4, 7] с применением различных теорий (битовых векторов, линейной арифметики и других). Эти инструменты позволяют не только выяснить, выполнима ли формула, но и подобрать значения переменных, для которых она выполняется, а в случае отсутствия решения предоставить набор ограничений, в которых заключено противоречие. Однако данная задача в общем случае является неразрешимой, вследствие чего решатели не всегда могут доказать наличие или отсутствие решения, а иногда и вовсе могут зависнуть. В случае существования решения ограничений для типов его всегда можно найти, однако доказательство его отсутствия по-прежнему остается неразрешимой задачей [3].

Для организации взаимодействия между теориями при решении ограничений в SMT-решателях используются специальные алгоритмы (например, DPLL(t)). В силу того, что решатель для типов не является SMT-теорией, по-умолчанию между ними отсутствует какое-либо взаимодействие. Однако информация, полученная во время решения ограничений на типы, может

оказаться полезной в случаях, когда SMT-решатель попытается приравнять какие-то адреса между собой, присвоить в какие-то из них null, а также в некоторых других. Её также можно использовать и в других компонентах символьной виртуальной машины.

Ранее в рамках учебных практик был разработан и реализован решатель для системы типов платформы .NET в проекте V#. Данная производственная практика нацелена на выстраивание взаимодействия между ним и SMT-решателем, а также другими компонентами символьной машины.

Постановка задачи

Целью производственной практики является увеличение тестового покрытия, получаемого при помощи символьного исполнения. Для достижения цели были поставлены следующие задачи:

- изучить работу SMT-решателя в комбинации теорий,
- разработать модель управления ограничениями,
- спроектировать процесс взаимодействия решателя для типов и SMT-решателя,
- реализовать разработанную систему,
- провести тестирование полученного решения.

1. Обзор

В данной главе рассмотрена предметная область данной производственной практике. В разд. 1.1 подробно рассмотрена работа символьного исполнения, разд. 1.2 посвящен SMT-решателям и их использованию в символьном исполнении, также в этом разделе обсуждается работа SMT-решателя в комбинации теорий. В разд. 1.3 показана архитектура проекта V#, в рамках которого выполняется данная производственная практика.

1.1 Символьное исполнение

Символьное исполнение — техника исследования программ, которая позволяет исполнять программный код в условиях неопределённости входных данных. Рассмотрим работу данной техники подробнее.

Изначально входные данные заменяются *символьными переменными* — абстракцией, позволяющей исследовать программу на всей совокупности значений входных данных сразу, а не на каких-то одних. Далее код программы выполняется последовательно по одной инструкции. Во время исполнения накапливается *условие пути pc* (англ. path condiditon) — формула логики первого порядка, отвечающая текущей ветви исполнения и ограничивающая возможные значения символьных переменных в ней. Изначально $pc = \top$. Также во время исполнения поддерживается *символьная память*. Она отвечает за хранение и модификацию символьных значений. При достижении ветвления символьная память и условие пути копируются столько раз, сколько ветвей может образоваться после ветвления. Далее каждой из ветвей в соответствующее условие пути добавляется условие попадания в нее. После модификации выясняется его выполнимость, что равносильно достижимости ветви, с помощью SMT-решателей и решателей для типов, которые будут рассмотрены позже. Затем продолжается независимое исследование достижимых ветвей.

```

public static int Abs(int x)
{
    int y = 0;
    if (x < 0) {
        y = -x;
    } else {
        y = x;
    }
    if (y < 0) {
        throw NegativeResultException();
    }
    return y;
}

```

Листинг 1: Программа для иллюстрации символьного исполнения

Рассмотрим работу символьного исполнения программы представленной на лист. 1.

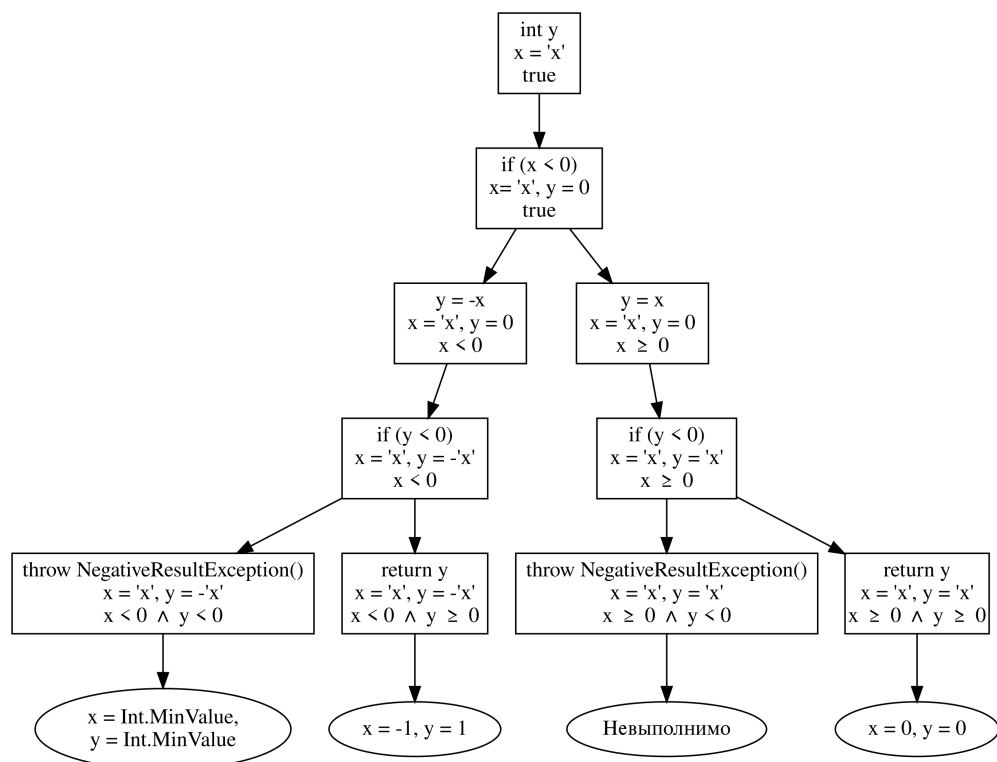


Рис. 1: Схема символьного исполнения функции Abs

На рис. 1 представлена схема символьного исполнения для программы

с лист. 1. В прямоугольниках на первой строке указана исполняемая инструкция, на второй — состояние символьной памяти, а на третьей — условие пути.

1.2 SMT-решатели

SMT-решатели (Satisfiability Modulo Theories) — это класс программных инструментов, которые автоматически решают задачу выполнимости булевых формул с добавлением теорий. SMT-решатели широко применяются в различных областях, таких как верификация программного обеспечения, анализ безопасности, автоматическое доказательство теорем и многих других. SMT-решатели могут работать с различными теориями, такими как целочисленная арифметика, линейная арифметика, теория массивов, теория списков и другие. В контексте символьного исполнения SMT-решатель является одним из решателей, используемых для проверки достижимости исследуемой ветви программы. Рассмотрим подробнее результаты работы SMT-решателя.

SAT Если SMT-решателю удалось найти *модель* — набор значений переменных, обращающих входную формулу в истину.

UNSAT Если SMT-решателю удалось доказать отсутствие решения, что равносильно противоречивости входной формулы. Также некоторые SMT-решатели в данном случае возвращают *ядро* (англ. UNSAT core) — минимальный набор конъюнктов, содержащих противоречие.

UNKNOWN В силу неразрешимости теорий или некоторых их комбинаций, SMT-решатели могут зависнуть в процессе своей работы. Поэтому в случае, когда время работы SMT-решателя превысило допустимый предел, возвращается такой результат.

Рассмотрим DPLL(T) — механизм работы SMT-решателя в комбинации теорий, использующийся в таких решателях, как Z3 Theorem Prover, CVC4. Во время работы этого алгоритма входная формула преобразуется в SAT-формулу путём замены частей, относящихся к теориям, в булевы переменные. Решается задача SAT, после чего происходит опрос решателей

теорий. Если формула оказалась выполнимой, процесс завершается, иначе же исходная формула дополняется информацией из UNSAT-ядер решателей теорий, и процесс повторяется.

1.3 Проект V#

V#¹ — инструмент для анализа программ, он исследует CIL (Common Intermediate Language) с использованием техники символьного исполнения. Далее будет рассмотрена архитектура проекта V# (рис. 2). Цветом выделена компонента, модифицированная в рамках данной работы.

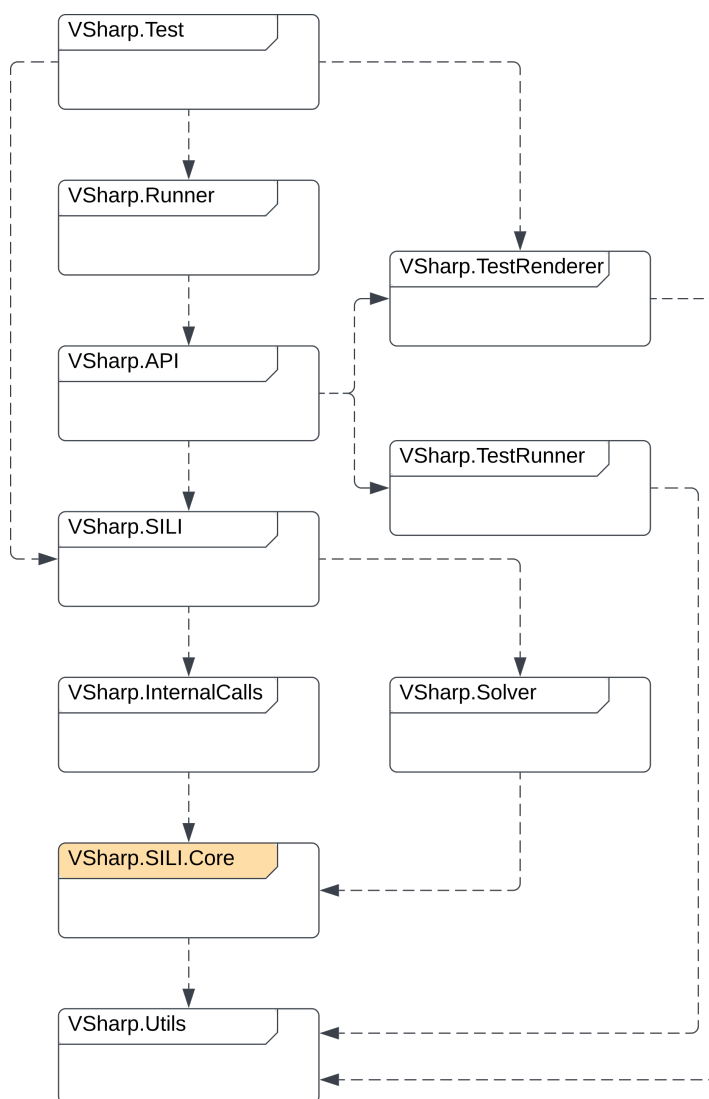


Рис. 2: Архитектура проекта V#

Рассмотрим каждую компоненту подробнее.

¹<https://github.com/VSharp-team/VSharp/>

VSharp.Test отвечает за проведение интеграционного тестирования.

VSharp.Runner предоставляет CLI (Command Line Interface) для использования V# через терминал.

VSharp.API предоставляет интерфейс для использования V# из других проектов.

VSharp.TestRenderer занимается генерацией кода тестов на языке C#.

VSharp.TestRunner используется для запуска сгенерированных тестов.

VSharp.SILI (Symbolic Intermediate Language Interpreter) является символьным интерпретатором CIL.

VSharp.InternalCalls содержит символьные модели некоторых функций из CLR (Common Language Runtime), что позволяет вместо полноценного их исследования подставить готовые модели.

VSharp.Solver реализует систему взаимодействия с SMT-решателем.

VSharp.SILI.Core представляет из себя «ядро» проекта. В нем происходят все нетривиальные операции внутри символьной машины, например, работа с символьной памятью, решение ограничений на типы.

VSharp.Utils содержит вспомогательные функции для остальных компонент.

Список литературы

- [1] Cadar Cristian and Sen Koushik. Symbolic execution for software testing: three decades later // *Communications of the ACM*. — 2013. — Vol. 56, no. 2. — P. 82–90.
- [2] De Moura Leonardo and Bjørner Nikolaj. *Z3: An efficient SMT solver* // International conference on Tools and Algorithms for the Construction and Analysis of Systems / Springer. — 2008. — P. 337–340.
- [3] Misonizhnik Aleksandr and Mordvinov Dmitry. On Satisfiability of Nominal Subtyping with Variance // 33rd European Conference on Object-Oriented Programming (ECOOP 2019) / Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. — 2019.
- [4] Niemetz Aina and Preiner Mathias. Bitwuzla at the SMT-COMP 2020 // arXiv preprint arXiv:2006.01621. — 2020.
- [5] Păsăreanu Corina S and Visser Willem. A survey of new trends in symbolic execution for software testing and analysis // *International journal on software tools for technology transfer*. — 2009. — Vol. 11, no. 4. — P. 339–353.
- [6] Baldoni Roberto, Coppa Emilio, D’elia Daniele Cono, Demetrescu Camil, and Finocchi Irene. A survey of symbolic execution techniques // *ACM Computing Surveys (CSUR)*. — 2018. — Vol. 51, no. 3. — P. 1–39.
- [7] Barrett Clark, Conway Christopher L, Deters Morgan, Hadarean Liana, Jovanović Dejan, King Tim, Reynolds Andrew, and Tinelli Cesare. *Cvc4* // International Conference on Computer Aided Verification / Springer. — 2011. — P. 171–177.