Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Кафедра системного программирования

Черниговская Лидия Александровна

Реализация синхронизирующего решателя дизъюнктов Хорна

Выпускная квалификационная работа

Научный руководитель:
ст. преп. каф. СП Я. А. Кириленко

Консультант:
Д. А. Мордвинов

Рецензент:
к.т.н. Федюкович Г. Г.

Санкт-Петербург
2018

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software Engineering

Lidiia Chernigovskaia

Implementation of the synchronizing constrained Horn solver

Graduation Project

Scientific supervisor:
Iakov Kirilenko

Consultant:
Dmitry Mordvinov

Reviewer:
Postdoctoral Research Associate Grigory Fedyukovich

Saint-Petersburg
2018

Оглавление

Введение

Компьютеры стали незаменимой частью современного мира и контролируют все большее количество его аспектов. С учетом этого вопросы корректности программ выходят на первый план. Наряду с традиционной ручной разработкой тестов существуют подходы по *автоматической генерации тестового покрытия*, *формальной верификации* и *синтеза* программного кода.

С середины 2000-х годов в контексте всех трех направлений особую популярность приобрели системы автоматического доказательства теорем для задачи SMT[1] — *SMT-решатели* [4, 7, 8, 19]. Задача SMT заключается в том, чтобы определить выполнима ли формула *языка первого порядка*, где функции и предикатные символы интерпретированы согласно заранее заданным *теориям*. Другими словами, задача SMT аналогична задаче выполнимости булевых формул (задаче SAT), но булевы переменные обобщены до предикатов заранее заданных теорий, а также возможно появление кванторов. Например, линейные неравенства над вещественными или целочисленными переменными ($2x + 4y \geq 6$) являются предикатами множества теории линейной вещественной или целочисленной арифметики и вычисляются согласно правилам этих теорий. Помимо теорий линейной целочисленной и вещественной арифметики, SMT-решатели могут поддерживать теории массивов и алгебраических типов данных, теории битовых векторов, строк, неинтерпретируемых функций. SMT-решатели эффективно комбинируют алгоритмы решения задачи SAT (как, к примеру, DPLL [6]) с разрешающими процедурами теорий.

Несмотря на все преимущества, SMT-решатели плохо подходят для рассуждения о рекурсивных программах. Доказательство безопасности программы, как правило, опирается на поиск *безопасного индуктивного инварианта*. Для программы $< Init, Tr, P >$, где $Init$ — начальное состояние, Tr — отношение перехода программы, а P — свойство безопасности программы, безопасный индуктивный инвариант Inv должен

[1] Satisfiability Modulo Theories — задача выполнимости формул в теориях

удовлетворять следующим 3 свойствам:

$$\begin{aligned} \forall u \in S \quad & \big(Init(u) \Rightarrow Inv(u)\big) \\ \forall u, v \in S \quad & \big(Inv(u) \wedge Tr(u, v) \Rightarrow Inv(v)\big) \\ \forall u \in S \quad & \big(Inv(u) \Rightarrow P(u)\big), \end{aligned}$$

где S — множество состояний программы.

Проблема заключается в том, что задача поиска безопасного индуктивного инварианта для рекурсивных программ не выражается в логике первого порядка.

Тем не менее в последние пять лет активно развивается подход, вдохновленный логическим программированием [12], в котором программа описывается в виде набора ограниченных дизъюнктов Хорна — формул логики первого порядка особого вида, где помимо интерпретированных функциональных и предикатных символов имеется множество неинтерпретированных реляционных символов (формальное определение дано в главе 2). Инструменты, решающие такие системы (далее — *Хорн-решатели*), используют всю инфраструктуру SMT-решателей и современные подходы, использующиеся для верификации программного кода [5, 10, 11, 14, 21]. В случае, когда Хорн-решатель завершается на системе ограниченных дизъюнктов Хорна, результатом его работы будет служить либо доказательство выполнимости системы дизъюнктов Хорна (индуктивный инвариант безопасности), либо контрпример в виде дерева вывода.

Возможна, однако, ситуация, когда индуктивный инвариант безопасности системы не выражается в теориях Хорн-решателей, и процесс поиска решения не завершается. Ценными в таком случае являются методы *ослабления* индуктивного инварианта. Недавно в статье [15] была предложена операция *синхронного произведения* ограниченных дизъюнктов Хорна, которая во многих случаях трансформирует систему ограниченных дизъюнктов Хорна таким образом, что не решающиеся современными Хорн-решателями системы начинают решаться за доли секунды.

Алгоритм, представленный в упомянутой статье [15], был реализован и протестирован на базе системы Rosette [22]. Система Rosette не так популярна, как Хорн-решатели Z3 [7] или SPACER [1, 14], и, т.к. она является «надстройкой» над ними, реализация алгоритма синхронизации на уровне Rosette не позволяет получить всех преимуществ от тесной интеграции с ядром Z3, как в случае расширения кодовой базы непосредственно Z3.

Поздние эксперименты показали, что возможно улучшить алгоритм синхронного произведения с помощью вычисления *синхронизационных лемм* — индуктивных утверждений, описывающих связь двух или более цепочек рекуррентных вычислений. Задача синхронизационных лемм состоит в определении необходимости проведения операции синхронного произведения, а также в облегчении поиска безопасного индуктивного инварианта.

1. Постановка задачи

Целью данной работы является реализация улучшенного алгоритма синхронного произведения ограниченных дизъюнктов Хорна для Хорн-решателей Z3 и SPACER.

Были поставлены следующие задачи.

1. Изучить и реализовать алгоритм синхронного произведения дизъюнктов Хорна на базе Z3 и SPACER.

2. Предложить алгоритм поиска синхронизационных лемм и реализовать его.

3. Провести эксперименты.

2. Обзор

В последние пять лет появился целый класс верификаторов [13, 16, 20], использующих Хорн-решатели. Наиболее известные Хорн-решатели — Z3 [10], SPACER [1], ELDARICA [23], CVC4 [4].

В обзоре будут даны основные понятия, используемые в контексте Хорн-решателей, будет рассказано о Хорн-решателях Z3 и SPACER, так как данная работа представляет расширение кодовой базы ядра Z3, и о наиболее близком к теме диплома проекте VeriMAPRel.

Про алгоритм синхронного произведения дизъюнктов Хорна, на который опирается данная работа, будет рассказано в следующей главе.

2.1. Основные определения

Пусть имеются два непересекающиеся множества функциональных и предикатных символов $\mathcal{F}$ и $\mathcal{P}$, а также функция ar, которая сопоставляет данные символы их арности. Интерпретация этих символов задается структурой $(\mathcal{D}, \sigma)$ с непустым доменом $\mathcal{D}$. Для каждой n-арной функции $f \in \mathcal{F}$ имеем $\sigma(f) : \mathcal{D}^n \to \mathcal{D}$ и для каждого n-арного предиката $p \in \mathcal{P}$, $\sigma(p) \subseteq \mathcal{D}^n$. $\mathcal{X}$ — счетное множество переменных. Язык первого порядка $\mathcal{A}$ бесквантворных формул над $\mathcal{F}$, $\mathcal{P}$, и $\mathcal{X}$ называется *языком ограничений*, формула из $\mathcal{A}$ — *ограничением*.

Пусть также имеется фиксированное множество *неинтерпретированных реляционных символов* $\mathcal{R}$, такое что $\mathcal{R} \cap (\mathcal{F} \cup \mathcal{P}) = \emptyset$.

Определение 1. *Ограниченный дизъюнкт Хорна* — формула C в логике первого порядка, имеющая следующий вид:

$$\phi \wedge r_1(\vec{x_1}) \wedge \ldots \wedge r_k(\vec{x_k}) \Rightarrow H,$$

где $r_i \in \mathcal{R}$, а ϕ является ограничением. H — это либо атом $p_0(\vec{x_0})$, где $p_0 \in \mathcal{R}$, либо $\perp$ (ложь). $\vec{x_i}$ — вектор переменных, $\vec{x_i}[j] \in \mathcal{X}$ для любого $0 \leq j < |\vec{x_i}|$.

Посылку импликации мы будем называть *телом* дизъюнкта, а за-

ключение — его *головой*:

$$body(C) \stackrel{\text{def}}{=} \{\phi, p_1(\vec{x_1}), \ldots, p_k(\vec{x_k})\} \qquad\qquad head(C) \stackrel{\text{def}}{=} H$$

Будем выделять три части тела дизъюнкта: $body(C) = \{\phi\} \cup L \cup R$, где ϕ — ограничение, R — множество *рекурсивных* вызовов: $R = \{r \in body(C) \mid rel(r) = rel(head(C))\}$, где $rel(r)$ возвращает неинтерпретированный реляционный символ, соответствующий атому r, и L — множество *нерекурсивных* вызовов.

Определение 2. *Система ограниченных дизъюнктов Хорна* — это пара (P, q), где P — множество ограниченных дизъюнктов Хорна, а $q \in P$ — дизъюнкт, называемый запросом, такой, что $head(q) = \bot$. Ограниченные дизъюнкты Хорна из (P, q) будем называть *правилами.*

Системы ограниченных дизъюнктов Хорна решаются с помощью *Хорн-решателей.* Если Хорн-решатель завершается на системе ограниченных дизъюнктов Хорна, результатом его работы является ответ — выполнима данная система или нет. Поясним это более формально.

Определение 3. Система ограниченных дизъюнктов Хорна называется *выполнимой,* если существует отображение $I : \mathcal{R} \to \mathcal{A}$, такое что $\bigwedge_{i=1}^{n} Cl_\forall(J(C_i))$ выполнимо. $Cl_\forall(\Phi)$ — *универсальное замыкание* Φ, C_i — ограниченные дизъюнкты Хорна. $J(C)$ определяется следующим образом:

$$J(C) \stackrel{def}{=} \left(\phi \wedge I(p_1)(\vec{x_1}) \wedge \ldots \wedge I(p_k)(\vec{x_k}) \Rightarrow \left\{ \begin{array}{r} I(p)(\vec{x}), \quad if\ head(C) = p(\vec{x}) \\ \bot, \quad if\ head(C) = \bot \end{array} \right)\right.$$

I задает интерпретацию неинтерпретированным символам, является *решением* системы и называется *безопасным индуктивным инвариантом системы.*

Определение 4. Система ограниченных дизъюнктов Хорна является *невыполнимой,* если существует такой набор переменных, что $body(q)$ — выполнимо. Решением в данном случае является дерево достижимости $body(q)$.

Определение 5. *Правила,* соответствующие определенному реляционному символу, будем определять следующим образом:

$$rules(p) \stackrel{def}{=} \{C \in P \mid rel(head(C)) = p\},$$

где P — система ограниченных дизъюнктов Хорна.

2.2. Проект Z3

Z3 — система автоматического доказательства теорем, разрабатываемая Microsoft Research. В Z3 осуществлена поддержка теорий неинтерпретируемых функций, арифметики, битовых векторов, массивов, алгебраических типов данных. Имеется возможность комбинирования данных теорий.

В Z3 реализован подход *GPDR*[2] [10], адаптирующий алгоритм верификации булевых программ PDR[3] [2] для решения систем ограниченных дизъюнктов Хорна в контексте SMT-теорий.

2.3. Проект SPACER

Проект SPACER[4] является ответвлением от Z3 и использует его инфраструктуру.

Особенностью SPACER является алгорим, итеративно строящий одновременно переапроксимацию программы (для поиска инварианта) и ее недоапроксимацию для поиска контрпримера. На сегодняшний день SPACER является одним из самых конкурентоспособных решателей дизъюнктов Хорна.

2.4. Проект VeriMAPRel

Одной из идей логического программирования является синхронизация рекуррентных вычислений с помощью правил *folding*/*unfolding* [3,

[2]Generalized Property Directed Reachability

[3]Property Directed Reachability

[4]https://bitbucket.org/spacer/code

9]. Недавно эта идея была применена в контексте реляционной верификации программ [3, 18]. Вместе с шагами `fold` и `unfold` эти работы *специализируют* дизъюнкты для "проталкивания" ограничений внутрь рекурсивных вычислений [17].

Синхронное произведение дизъюнктов Хорна [15], на которое опирается данная дипломная работа, не зависит от правил folding/unfolding и на практике создает более компактные системы.

3. Алгоритм синхронного произведения ограниченных дизъюнктов Хорна

Данная глава посвящена алгоритму синхронного произведения, предложенному в статье [15]. В рамках дипломной работы предложенный алгоритм был реализован на базе Z3.

Для начала дадим необходимые определения. Для краткости в дальнейшем тексте данной работы конъюнкция будет обозначаться запятой, а импликация — $\leftarrow$.

Определение 6**.** Будем говорить, что непустые множества $A_1, \ldots, A_n$ *покрываются* A (обозначим как $A \in \lfloor A_1, \ldots, A_n \rfloor$), если:

1. $A \subseteq A_1 \times \ldots \times A_n$;
2. каждый элемент из множеств A_i при построении A был использован хотя бы один раз, т.е. находится на i-ой позиции как минимум в одном наборе $x \in A$.

Пример 1**.** Для множества натуральных чисел справедливо следующее:

$$\{(1,3,5),(2,4,5)\} \in \lfloor \{1,2\},\{3,4\},\{5\} \rfloor$$
$$\{(1,3,5),(2,3,5)\} \notin \lfloor \{1,2\},\{3,4\},\{5\} \rfloor$$

Определение 7**.** Имеются предикаты $p_1, \ldots, p_n$ и r такие, что $ar(r) = \sum_{i=1}^{n} ar(p_i)$. Обозначим за $\prod_{i=1,r}^{n} p_i(\vec{x_i})$ выражение $r(\vec{x_1} \cdot \ldots \cdot \vec{x_n})$, где $\vec{x} \cdot \vec{y}$ — конкатенация $\vec{x}$ и $\vec{y}$.

Если из контекста понятны индексы, будем их опускать и писать $\prod_{r} p_i(\vec{x_i})$.

Определение 8**.** C — ограниченный дизъюнкт Хорна, $body(C) = \{\phi\} \cup L \cup R$. Множество $R_{/head}(C)$ определим как:

$$R_{/head}(C) \stackrel{def}{=} \begin{cases} head(C), & \textit{если } R = \emptyset \\ R, & \textit{в противном случае} \end{cases}$$

Таким образом, если вместо R использовать $R_{/head}$, правила вида $p(\vec{x}) \leftarrow \phi$ преобразуются в $p(\vec{x}) \leftarrow \phi, p(\vec{x})$.

Определение 9**.** Пусть $C_1, \ldots, C_n$ — ограниченные дизъюнкты Хорна, $p_1, \ldots, p_n$ различные реляционные символы, такие, что $C_i \in rules(p_i)$. p — новый реляционный символ, $p \notin \mathcal{R}$ и $ar(p) = \sum\limits_{i=1}^{n} ar(p_i)$. C над $\mathcal{R} \cup \{p\}$ называется *произведением* $C_1, \ldots, C_n$ *относительно* p, обозначается $C = C_1 \times_p \ldots \times_p C_n$, если:

$$head(C) = \prod_p head(C_i); \qquad body(C) = \{\phi\} \cup L \cup R \setminus \{head(C)\};$$

$$\text{где:} \quad \phi = \bigwedge_{i=1}^{n} \phi_i; \quad L = \bigcup_{i=1}^{n} L_i;$$

$$R = \left\{ \prod_p r_i \;\middle|\; (r_1, \ldots r_n) \in \left\lfloor R_{/head}(C_1), \ldots, R_{/head}(C_n) \right\rfloor \right\}$$

Пример 2**.** Пусть имеется набор из двух ограниченных дизъюнктов Хорна:

$$C_1 : f(x_1, x_2) \leftarrow x_3 = x_2 + 1, x_4 = x_2 + 2, f(x_1, x_3), f(x_1, x_4),$$
$$C_2 : g(y) \leftarrow y > 0,$$

где f и g — реляционные символы, $x_3 = x_2 + 1, x_4 = x_2 + 2$ и $y > 0$ — ограничения ϕ_1 и ϕ_2; L_i пустые; $R_{/head}(C_1) = \{f(x_1, x_3), f(x_1, x_4)\}$, $R_{/head}(C_2) = \{g(y)\}$.

Введем новый реляционный символ fg. Таким образом для $C_1 \times_{fg} C_2$ имеем:

$$fg(x_1, x_2, y) \leftarrow x_3 = x_2 + 1, x_4 = x_2 + 2, y > 0, fg(x_1, x_3, y), fg(x_1, x_4, y)$$

Определение 10**.** $P = \{C_1, \ldots, C_m\}$ набор ограниченных дизъюнктов Хорна над $\mathcal{R}$, $p_1, \ldots, p_n$ — реляционные символы из $\mathcal{R}$. *Произведение реляционных символов* $p_1, \ldots, p_n$ (обозначим как: $p_1 \times \ldots \times p_n$) — это пара $(p, rules(p))$, где p новый реляционный символ, такой, что: $ar(p) = \sum\limits_{i=1}^{n} ar(p_i)$, и $rules(p) = \{C_1 \times_p \ldots \times_p C_n \mid (C_1, \ldots, C_n) \in rules(p_1) \times \ldots \times$

$rules(p_n)\}$.

Пример 3**.** Добавим к примеру 2 ещё одно правило:

$$\begin{aligned} f(x_1, x_2) &\leftarrow x_3 = x_2 + 1, x_4 = x_2 + 2, f(x_1, x_3), f(x_1, x_4), \\ g(y) &\leftarrow y > 0, \\ g(y_1) &\leftarrow y_2 = y_1 - 1, g(y_2). \end{aligned}$$

Произведение реляционных символов f и g:

$$\begin{aligned} fg(x_1, x_2, y) &\leftarrow x_3 = x_2 + 1, x_4 = x_2 + 2, y > 0, fg(x_1, x_3, y), fg(x_1, x_4, y) \\ fg(x_1, x_2, y_1) &\leftarrow x_3 = x_2 + 1, x_4 = x_2 + 2, y_2 = y_1 - 1, \\ &\qquad fg(x_1, x_3, y_2), fg(x_1, x_4, y_2) \end{aligned}$$

Определение 11**.** Пусть $P = \{C_1, \ldots, C_m\}$ — набор ограниченных дизъюнктов Хорна, а некоторый дизъюнкт $C_a \in P$ имеет следующий вид:

$$C_a = H \leftarrow \phi, p_1(\vec{x_1}), \ldots, p_n(\vec{x_n}), W$$

где для любых $1 \leq i \leq n$: $p_i \neq rel(H)$, а W может содержать другие применения реляционных символов из $\mathcal{R}$. $(p, rules(p)) = p_1 \times \ldots \times p_n$. *Синхронизация* P — это новый набор дизъюнктов Хорна P' над $\mathcal{R}' = \mathcal{R} \cup \{p\}$, полученный из P добавлением новых правил и заменой применений p_i в C_a на их произведение

$$P' \stackrel{def}{=} P \setminus \{C_a\} \cup rules(p) \cup \{C'_a\},$$

где C'_a имеет вид:

$$C'_a \stackrel{def}{=} H \leftarrow \phi, \prod_p p_i(\vec{x_i}), W$$

Пример 4**.** Добавим к набору правил из примера 3 следующее правило:

$$\bot \leftarrow a = c, f(a, b), g(c)$$

После синхронизации из опр. 11 данное правило будет заменено на

$$\bot \leftarrow a = c, fg(a, b, c)$$

Алгоритм 1: Синхронное произведение дизъюнктов Хорна

Входные данные: система ограниченных дизъюнктов Хорна (P, q)

Выходные данные: трансформированная система (P', q')

Исходные параметры: набор ограниченных дизъюнктов Хорна WL, кэш новых реляционных символов $cache$

```
1  P' ← ∅;
2  cache ← ∅;
3  WL ← P;
4  while (¬EMPTY(WL)) do
5      C_a ← GET(WL);
6      NonRec ← GETREC(GETNONRECPART(C_a));
7      if NonRec >= 2 and HASNOT(cache, p) then
8          foreach ({p_1(x_1), ..., p_n(x_n)} ∈ NonRec) do
9              (p, rules(p)) ← p_1 × ... × p_n;
10             cache ← cache ∪ p;
11             C_a ← REPLACE(C_a, p_1(x_1), ..., p_n(x_n), ∏_p p_i(x_i));
12             foreach (C ∈ rules(p)) do
13                 WL ← WL ∪ C
14             end
15         end
16     end
17     P' ← P' ∪ C_a;
18 end
```

Перейдем к самому алгоритму. На вход подается система ограниченных дизъюнктов Хорна. Имеется рабочий набор дизъюнктов WL, изначально равный множеству дизъюнктов, поступивших на вход. Эти дизъюнкты добавляются в новую систему либо трансформированными, либо без изменений. Трансформация происходит над теми дизъюнктами C_a, которые имеют два и более вызова реляционных символов $p_i \in L(C_a)$, при этом для каждого такого p_i, $rules(p_i)$ должно содержать хотя бы одно *рекурсивное* правило — ограниченный дизъюнкт Хорна, у которого множество рекурсивных вызовов R не пусто. Затем происходит синхронизация, описанная в определении 11. Новые правила, полученные на данном шаге, добавляются в рабочий набор правил.

4. Алгоритм поиска синхронизационных лемм

Операция синхронного произведения дизъюнктов Хорна (алг. 1) производится для дизъюнктов Хорна, имеющих в теле два или более нерекурсивных применения рекурсивных символов. Однако, синхронизация — достаточно дорогая операция, порождающая большое количество новых правил, и синхронизировать *все* рекурсивные вычисления в программе достаточно расточительно. Важно, что синхронизация должна связывать не все пары рекурсивных вычислений, а только те, что имеют некоторое *сходство* (например, оперируют одними и теми же данными).

Для того, чтобы понять, когда требуется синхронизировать вызовы, а когда не требуется, было предложено искать *логические соотношения между аргументами* синхронизируемых правил. Данные соотношения должны быть *индуктивными*, т.е. должны быть верны на протяжении всего синхронного вычисления. Будем называть такие утверждения *синхронизационными леммами* и обозначать буквой ρ.

Определим синхронизационную лемму более формально.

Определение 12**.** $P = \{C_1, \ldots, C_m\}$ — набор ограниченных дизъюнктов Хорна, $C_a \in P$ следующего вида:

$$C_a = H \leftarrow \phi, p_1(\vec{x_1}), \ldots, p_n(\vec{x_n}), W$$

для любых $1 \leq i \leq n$: $p_i \neq rel(H)$, и $rules(p_i)$ содержат рекурсивные правила. $p_1(\vec{x_1}), \ldots, p_n(\vec{x_n})$ в данном случае являются *синхронизируемыми вызовами.* Процесс вычислений начинается с правила C_a.

Синхронизационная лемма ρ должна сопоставлять аргументы синхронизируемых вызовов и выполняться при ограничении C_a, то есть:

$$\rho(\vec{x_1} \cdot \ldots \cdot \vec{x_n}) \leftarrow \phi$$

Синхронизационная лемма ρ должна быть индуктивна относительно отношения перехода синхронизации $p_1, \ldots, p_n$.

В подразделах 4.2, 4.3 описаны предложенные и реализованные на базе Z3 алгоритмы поиска синхронизационных лемм. В подразделе 4.1

вводится понятие графа достижимости правил, необходимое для лучшего понимания работы представленных алгоритмов.

4.1. Граф достижимости правил

По исходной системе ограниченных дизъюнктов Хорна (P, q) строится ориентированный граф достижимости правил. Построение графа начинается с правила запроса.

Опишем алгоритм построения графа достижимости для одного узла. Пусть хотим узнать, в какие правила мы можем попасть из C : $H \leftarrow \phi, r_1(\vec{x_1}), \ldots, r_n(\vec{x_n})$. Для каждого неинтерпретированного символа r_i рассматриваем набор правил $rules(r_i)$, соответствующий ему. Это те правила, в которые мы теоретически можем попасть из C. Для каждого правила из $rules(r_i)$ производится унификация с $r_i(\vec{x_i})$. Если подстановка правила вместо $r_i(\vec{x_i})$ возможна, то она производится, и проверяется ограничение полученного после подстановки правила. Если подстановка не нарушает истинности ограничения, то правило объявляется достижимым из дизъюнкта C, и в графе достижимости добавляется ребро.

Граф синхронного произведения дизъюнктов Хорна — прямое произведение исходных графов.

Затем производится расслаивание полученного графа достижимости правил на компоненты сильной связности.

Будем говорить, что компонента сильной связности *соответствует* реляционному символу r_i, если среди правил, входящих в данную компоненту, имеется правило, в голове которого стоит реляционный символ r_i. В случае взаимно-рекурсивных правил компонента сильной связности может соответствовать нескольким реляционным символам.

4.2. Первый подход

Вычисление начинается с правила $C_a = H \leftarrow \phi, p_1(\vec{x_1}), \ldots, p_n(\vec{x_n}), W$, ребрам в графе синхронного произведения, выходящим из вершины, соответствующей C_a, сопоставим ϕ. Необходимо посчитать леммы для

всех вершин, соответствующих произведению $p_1, \ldots, p_n$. Сначала посчитаем синхронизационную лемму для вершины, в которую входит только ребро из вершины, соответствующей C_a. В качестве исходной леммы берем ϕ, затем исключаем конъюнкты, нарушающие индуктивность леммы с помощью алгоритма 2. После того, как посчитана лемма ρ для этой вершины, всем выходящим из нее ребрам сопоставим лемму ρ. Продолжим вычисление синхронизационных лемм для вершин, соответствующих произведению $p_1, \ldots, p_n$. Для того, чтобы посчитать лемму для правил, соответствующих этой вершине, рассмотрим ребра, входящие в нее. В качестве исходной леммы возьмем дизъюнкцию лемм, соответствующих этим ребрам, в КНФ. С помощью алгоритма 2 посчитаем лемму ρ_1 и сопоставим ее выходящим из вершины ребрам.

Опишем алгоритм отбрасывания конъюнктов.

На вход поступает лемма, состоящая из m конъюнктов, проверим, какие из этих конъюнктов индуктивны. Для корректной работы алгоритма переименовываем переменные в дизъюнктах, для которых считается синхронизационная лемма, таким образом, что если переменная входит в какой-то дизъюнкт C_i, она не может входить ни в какой другой дизъюнкт C_j, где $j \neq i$. Затем сохраняем атомы, стоящие в голове этих правил, а также атомы из множества $R_{/head}$ (строки 5, 8, 9), для дальнейшего применения на них синхронизационной леммы в строках 12, 14.

Индуктивность леммы должна обеспечивать следующее: если выполнены ограничения правил $C_1 \ldots C_n$ и лемма выполняется для аргументов голов данных правил, то должен существовать такой набор свободных переменных, что лемма выполняется и для аргументов рекурсивных вызовов ($calls$). Таким образом, нам необходимо проверить выполнимость такой формулы:

$$Cl_{\exists}\Big(e_1\big(args(calls)\big), \ldots, e_m\big(args(calls)\big)\Big) \leftarrow \phi_1, \ldots, \phi_n, \rho\big(args(heads)\big) \quad (1)$$

Формула 1 истинна, если ложна следующая формула:

$$\begin{aligned} &\phi_1, \ldots, \phi_n, \rho(args(heads)), \\ &Cl_{\forall}\big(args(calls)\big)\big(\neg e_1(args(calls)) \vee \ldots \vee \neg e_m(args(calls))\big) \end{aligned} \quad (2)$$

Алгоритм 2: Алгоритм отбрасывания конъюнктов

Входные данные: исходная лемма $\rho(\vec{x_1}, \ldots, \vec{x_n}) = (e_1, \ldots, e_m)$, набор правил $C_1, \ldots, C_n$, для которых считается синхронизационная лемма

Выходные данные: синхронизационнная лемма ρ

Исходные параметры: *enabled* — какие конъюнкты на данный момент входят в лемму, CD — множество дизъюнктов

```
1  for i = 1 to m do
2      enabled[i] ← true;
3  end
4  for i = 1 to n do
5      renameBoundVars(C_i);
6  end
7  for i = 1 to n do
8      heads ← heads ∪ head(C_i);
9      calls ← calls ∪ R_{/head}(C_i)
10 end
11 conjunct ← φ_1, ..., φ_n;
12 conjunct ← conjunct ∧ subst(ρ, args(heads));
13 Φ ← Φ ∧ conjunct;
14 conclusions ← conclusions ∪ (subst(ρ, args(calls)));
15 while (true) do
16     for i = 1 to n do
17         if enabled[i] then
18             CD ← CD ∪ not(conclusions[i]);
19         end
20     end
21     Φ ← Φ ∧ (forAllBoundVar(or(CD))) ;
22     if Φ is satisfiable then
23         model ← getModel;
24         valuation ← eval(conclusions, model);
25         core ← getUnsatCore;
26         deactivate(enabled, core);
27     end
28     if Φ is unsatisfiable then
29         ρ ← lemma(ρ, enabled);
30         break;
31     end
32     Return(ρ)
33 end
```

Проверяем выполнимость последней формулы (строки 11— 21). Если формула 2 невыполнима, то выполнима формула 1, а значит лемма индуктивна, и алгоритм заканчивает работу(строка 22).

Если формула 2 *выполнима*, лемма, имеющаяся в данный момент, неиндуктивна. Необходимо понять, какие конъюнкты исключить из леммы. Формула выполнима, значит, имеется модель, при которой эта формула выполнима (строка 23). Сопоставим данную модель с $e_i(args(calls))$ (строка 24). В данной модели формула 1 ложна, значит, ложно заключение данной формулы в данной модели. С помощью метода *GetUnsatCore* получим конъюнкты леммы, которые приводят к тому, что заключение формулы 1 в данной модели ложно. Данные конъюнкты отбрасываются — помечаются неиспользуемыми в *enabled* (строка 26. Выполнимость формулы проверяется заново уже без этих конъюнктов.

Рассмотрим работу данного алгоритма на примере системы ограниченных дизъюнктов Хорна, описывающей монотонность произведения целых неотрицательных чисел.

Пример 5**.**

$$mult(x, y, z) \leftarrow x = 0, z = 0,$$
$$mult(x, y, z) \leftarrow x > 0, x' = x - 1, z = z' + y, mult(x', y, z'),$$
$$\bot \leftarrow x_1 > 0, x_2 > x_1, y > 0, z_2 < z_1, mult(x_1, y, z_1), mult(x_2, y, z_2).$$

Посчитаем синхронизационную лемму для синхронизации двух рекурсивных правил $mult$. Попасть в такую комбинацию правил можем либо из правила-запроса, либо из такой же комбинации. Следовательно, исходная лемма состоит из 4 конъюнктов, являющихся ограничением запроса: $\rho_0(x_1, y, z_1, x_2, y, z_2) = x_1 > 0, x_2 > x_1, y > 0, z_2 < z_1$.

Переименуем переменные. Таким образом правила, для которых считается лемма:

$$C_1 = mult(a_1, b_1, c_1) \leftarrow a_1 > 0, a_1' = a_1 - 1, c_1' = c_1 - b_1, mult(a_1', b_1, c_1');$$
$$C_2 = mult(a_2, b_2, c_2) \leftarrow a_2 > 0, a_2' = a_2 - 1, c_2' = c_2 - b_2, mult(a_2', b_2, c_2').$$

Затем сформируем множества $heads$ и $calls$:

$$heads = \{mult(a_1, b_1, c_1), mult(a_2, b_2, c_2)\};$$
$$calls = \{mult(a'_1, b_1, c'_1), mult(a'_2, b_2, c'_2)\}.$$

Добавим посылку, согласно формуле 1.

$$result = \{a_1 > 0, a'_1 = a_1 - 1, c'_1 = c_1 - b_1, a_2 > 0, a'_2 = a_2 - 1, c'_2 = c_2 - b_2,$$
$$x_1 > 0, x_2 > x_1, y > 0, z_2 < z_1,$$
$$a_1 = x_1, a_2 = x_2, b_1 = y, b_2 = y, c_1 = z_1, c_2 = z_2\}$$

И осталось сформировать последний конъюнкт для формулы 2:

$$\forall(a, b, c, a', c')\big(a \leq 0 \vee a' \leq a \vee b \leq 0 \vee c' \geq c \vee$$
$$\vee a'_1 \neq a \vee a'_2 \neq a' \vee b_1 \neq b \vee b_2 \neq b \vee c'_1 \neq c \vee c'_2 \neq c'\big)$$

Таким образом мы сформировали формулу 2. Данная формула выполнима. После подстановки модели в конъюнкты с заключениями, получим следующие утверждения:

$$a > 0, a' > a, b > 0, c' \leq c, 0 = a, 1 = b, 0 = c, 1 = a', -1 = c'$$

Решатель выдает $unsat$. Unsat ядро содержит $a > 0$ и $0 = a$. Данные атомы соответствовали в исходной лемме $\rho(x_1, y, z_1, x_2, y, z_2) = x_1 > 0, x_2 > x_1, y > 0, z_2 < z_1$ атому $x_1 > 0$, а также подстановке первого аргумента x_1. Таким образом, на следующем шаге конъюнкт с заключениями примет следующий вид:

$$\forall(a, b, c, a', c')\big(a' \leq a \vee b \leq 0 \vee c' \geq c \vee a'_1 \neq a \vee b_1 \neq b \vee b_2 \neq b \vee c'_1 \neq c \vee c'_2 \neq c'\big)$$

Решатель выдает $unsat$. Формула невыполнима, значит, искомая лемма имеет следующий вид:

$$\rho_1(x_1, y, z_1, x_2, y, z_2) = x_2 > x_1, y > 0, z_2 < z_1,$$

4.3. Второй подход

Идея, реализованная в предыдущем подходе, схожа с алгоритмом, описывающим работу решателя SPACER [1].

Подход, описанный в данном разделе основан на том, чтобы делегировать нахождение лемм самому Хорн-решателю. Для этого необходимо сформировать систему ограниченных дизъюнктов Хорна, описывающую индуктивность и безопасность синхронизационной леммы. Индуктивный инвариант данной системы будет являться искомой синхронизационной леммой.

Индуктивность леммы описывается с помощью правил, данных в определении 12.

Необходимо понять, как описать безопасность синхронизационной леммы (сформировать запрос).

В рамках данной работы в качестве запроса использовалась ситуация "рассинхронизации". Идея "рассинхронизации" заключается в том, что для набора синхронизируемых правил, для которых считается лемма, производится применение только одного правила из этого набора.

Пример 6***.*** *Пусть имеется такой набор синхронизируемых правил:*

$$\begin{aligned} f(x') &\leftarrow \varphi' \wedge f(x'') \\ g(y') &\leftarrow \psi' \wedge g(y'') \end{aligned}$$

Ситуация рассинхронизации для такого набора выражается в следующих двух правилах:

$$\begin{aligned} \bot &\leftarrow \varphi' \wedge \rho(x', y) \wedge \rho(x'', y) \\ \bot &\leftarrow \psi' \wedge \rho(x, y') \wedge \rho(x, y'') \end{aligned}$$

Если компонента сильной связности содержит правило запроса, то лемма, соответствующая ей, является ограничением правила запроса.

Опишем данный алгоритм подсчета синхронизационных лемм более формально.

Алгоритм 3: Алгоритм формирования системы дизъюнктов Хорна для поиска синхронизационной леммы

Входные данные: Компонента сильной связности K_i, для которой ищется лемма, уже найденные леммы для других компонент сильной связности $strata2lemmas$
Выходные данные: синхронизационнная лемма ρ
Исходные параметры: компоненты сильной связности $incomingStrata$, из которых достижима K_i

1 foreach $strata \in incomingStrata$ do
2 ADD($\rho \leftarrow strata2lemmas(strata)$)
3 end
4 foreach $mr \in MR_i$ do
5 foreach $recArgs \in RecArgs$ do
6 ADD($\rho(recArgs) \leftarrow \phi_1, \ldots, \phi_n, L(C_1), \ldots, L(C_n), \rho(args(head(C_1))) \cdot \ldots \cdot args(head(C_n)))$
7 end
8 foreach $C_i \in mr$ do
9 foreach $r_i \in R(C_i)$ do
10 ADD($\bot \leftarrow \phi_i, L(C_i), \rho(\vec{x_1} \cdot \ldots \vec{x_{i-1}} \cdot args\big(head(C_i)\big) \cdot \vec{x_{i+1}} \cdot \ldots \vec{x_n}), \rho(\vec{x_1} \cdot \ldots \vec{x_{i-1}} \cdot args(r_i) \cdot \vec{x_{i+1}} \cdot \ldots \vec{x_n})$)
11 end
12 end
13 end

Определение 13**.** $p_1, \ldots, p_n$ — реляционные символы, для которых требуется узнать синхронизационные леммы. Определим множество $K = \{K_i = (K_{i,1}, \ldots, K_{i,n}) \mid K_{i,j} \in S_j\}$, где S_j — множество компонент сильной связности, соответствующих p_j.

K_i достижима из K_j, если для $1 \leq m \leq n$ $K_{i,m}$ достижима из $K_{j,m}$

Определение 14**.** Определим множество комбинаций правил из K_i, как $MR_i = \{(C_1, \ldots, C_n) \mid C_j \in K_{i,j}\}$

Множество аргументов $RecArgs(C_1, \ldots, C_n) = \{args(r_1) \cdot \ldots \cdot args(r_n) \mid r_i \in (R_{/head}(C_i))\}$

Синхронизационная лемма считается для каждой компоненты связности из множества K, согласно алгоритму 3.

Для каждой входящей компоненты связности добавляем правила-факты для леммы (строка 2). Для всех комбинаций синхронизируемых правил из нашей компоненты сильной связности добавляем правило, обеспечивающего индуктивность синхронизационной леммы (строка 6). И для каждого синхронизируемого правила добавляем правило, соответствующее тому, что по этому правилу был совершен шаг, а по

остальным правилам нет, следовательно, аргументы, соответствующие этим правилам, остались без изменения (строка 10).

После формирования системы дизъюнктов Хорна, у решателя запрашивается выполнима ли данная система, если система невыполнима, то есть безопасна, индуктивный инвариант данной системы будет являться искомой синхронизационной леммой.

Рассмотрим пример формирования системы дизъюнктов Хорна для синхронизационной леммы.

***Пример* 7.** Рассмотрим систему, описывающую функцию div, возвращающую целую часть от деления. Данная система доказывает, что целая часть от деления числа a на y на 1 меньше, чем целая часть от деления $a + y$ на y:

$$\begin{aligned} div(x, y, 0) &\leftarrow x < y, \\ div(x', y, r') &\leftarrow x \geq y, x = x' - y, r' = r + 1, div(x, y, r), \\ &\bot \leftarrow y > 0, a > 0, r_1 \neq 1 + r, div(a, y, r), div(a + y, y, r_1). \end{aligned}$$

Хотим узнать, как связаны два вызова реляционного символа div в запросе — найти синхронизационную лемму для них. Реляционному символу div соответствуют две компоненты сильной связности, в каждой из которых по одному правилу:

$$\{div(x, y, 0) \leftarrow x < y\}, \{div(x', y, r') \leftarrow x \geq y, x = x' - y, r' = r + 1, div(x, y, r)\}$$

Возьмем для каждого применения div вторую компоненту связности и посчитаем синхронизационную лемму для $K_i = \big\{\{div(x'_1, y_1, r'_1) \leftarrow x_1 \geq y_1, x_1 = x'_1 - y_1, r'_1 = r_1 + 1, div(x_1, y_1, r_1)\}, \{div(x'_2, y_2, r'_2) \leftarrow x_2 \geq y_2, x_2 = x'_2 - y_2, r'_2 = r_2 + 1, div(x_2, y_2, r_2)\}\big\}$.

Компонента сильной связности K_i достижима из компоненты сильной связности, содержащей правило запроса, следовательно, первое правило для новой системы следующего вида(строка 2):

$$\rho(x, y, r, x', y, r') \leftarrow y > 0, a > 0, r' \neq 1 + r, x' = x + y$$

Затем добавляются правила, обеспечивающие индуктивность леммы при рекурсивном переходе (строка 6):

$$\begin{aligned}\rho(x_1', y_1, r_1', x_2', y_2, r_2') \leftarrow & \rho(x_1, y_1, r_1, x_2, y_2, r_2), \\ & r_2 = 1 + r_2', x_2 \geq y_2, x_2' = x_2 - y_2, \\ & r_1 = 1 + r_1', x_1 \geq y_1, x_1' = x_1 - y_1\end{aligned}$$

Правила, описывающие ситуацию "рассинхронизации"(строка 10):

$$\begin{aligned}\bot \leftarrow & \rho(x_1', y_1, r_1', x_2, y_2, r_2), \rho(x_1, y_1, r_1, x_2, y_2, r_2), \\ & r_1 = 1 + r_1', x_1 \geq y_1, x_1' = x_1 - y_1 \\ \bot \leftarrow & \rho(x_1, y_1, r_1, x_2', y_2, r_2'), \rho(x_1, y_1, r_1, x_2, y_2, r_2), \\ & r_2 = 1 + r_2', x_2 \geq y_2, x_2' = x_2 - y_2\end{aligned}$$

Отправив сформированную систему Хорн-решателю, получим индуктивный инвариант, который будет являться искомой синхронизационной леммой:

$$\rho(x, y, r, x', y', r') = x + y = x', (x + y) \leq (x' + y'), x + 2y = x' + y'$$

5. Реализация и эксперименты

Алгоритмы подсчета лемм и синхронного произведения ограниченных дизъюнктов Хорна были реализованы на уровне движка неподвижных точек Z3 (μZ). Язык разработки — C++.

Был реализован отдельный модуль трансформации, отвечающий за синхронизацию. Алгоритмы поиска синхронизационных лемм реализованы в различных ветках[5][6].

Для решения систем дизъюнктов Хорна в Z3 имеется возможность подключения различных опций. При решении системы Хорн-решатель обращается к доступным трансформациям в порядке приоритета. Опция, позволяющая подключать трансформацию, отвечающую за синхронизацию — fixedpoint.datalog.synchronization.

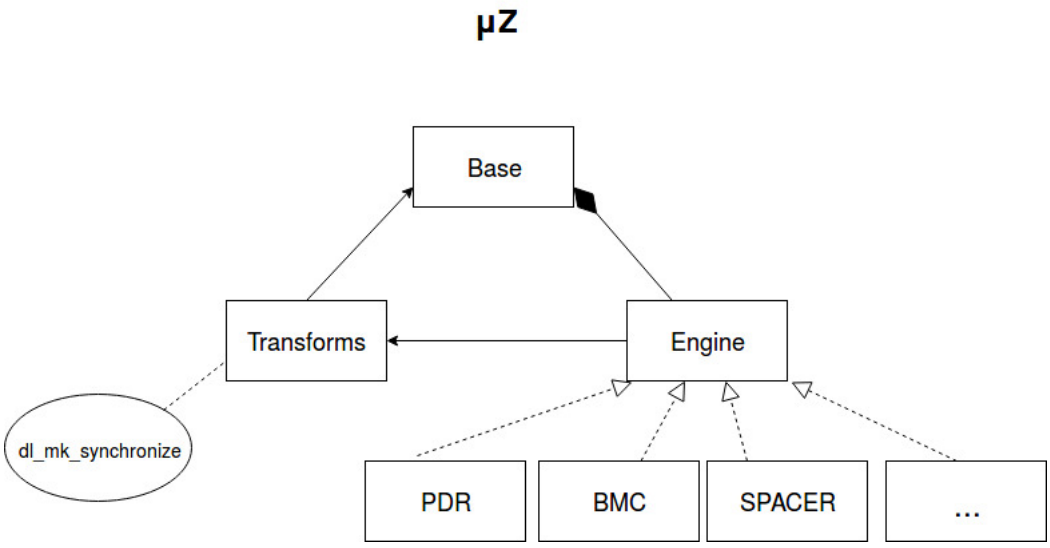

Рис. 1: Модуль неподвижных точек Z3

Для проведения экспериментов был сформирован набор нелинейных систем ограниченных дизъюнктов Хорна[7](в посылках дизъюнктов имеется больше одного рекурсивного вызова). Всего было сформировано 30 безопасных и 30 небезопасных тестовых программ. Результаты эксперимента приведены в таблице.

[5]https://github.com/LChernigovskaya/spacer/tree/lemma
[6]https://github.com/LChernigovskaya/spacer/tree/lemma_as_query
[7]https://github.com/LChernigovskaya/spacer/tree/lemma/benchmarks

		SPACER		SPACER + Synchronization	
Programs	Total	Solved	AvgTime	Solved	AvgTime
safety	30	6	0,081	25	0,27
unsafety	30	30	0,022	30	0,212

Заключение

В рамках данной работы было сделано следующее.

1. Изучен и реализован алгоритм синхронизации дизъюнктов Хорна для Хорн-решателей Z3 и SPACER.

2. Реализован алгоритм поиска синхронизационных лемм, основанный на отбрасывании конъюнктов.

3. Реализован алгоритм поиска синхронизационных лемм, основанный на формировании нового запроса к Хорн-решателю.

4. Поставлены эксперименты с решателем SPACER без синхронизации и с алгоритмом синхронного произведения, улучшенного с помощью синхронизационных лемм.

Список литературы

[1] Automatic abstraction in SMT-based unbounded software model checking / Anvesh Komuravelli, Arie Gurfinkel, Sagar Chaki, Edmund M Clarke // International Conference on Computer Aided Verification / Springer. — 2013. — P. 846–862.

[2] Bradley Aaron R. SAT-Based Model Checking without Unrolling. // Vmcai / Springer. — Vol. 6538. — 2011. — P. 70–87.

[3] Burstall Rod M, Darlington John. A transformation system for developing recursive programs // Journal of the ACM (JACM). — 1977. — Vol. 24, no. 1. — P. 44–67.

[4] CVC4 / Clark Barrett, Christopher L Conway, Morgan Deters et al. // International Conference on Computer Aided Verification / Springer. — 2011. — P. 171–177.

[5] Counterexample-guided abstraction refinement / Edmund Clarke, Orna Grumberg, Somesh Jha et al. // Computer aided verification / Springer. — 2000. — P. 154–169.

[6] DPLL (T): Fast decision procedures / Harald Ganzinger, George Hagen, Robert Nieuwenhuis et al. // CAV / Springer. — Vol. 4. — 2004. — P. 175–188.

[7] De Moura Leonardo, Bjørner Nikolaj. Z3: An efficient SMT solver // Tools and Algorithms for the Construction and Analysis of Systems. — 2008. — P. 337–340.

[8] De Moura Leonardo, Bjørner Nikolaj. Applications of SMT solvers to Program Verification // Notes for the Summer School on Formal Techniques. — 2014.

[9] Etalle Sandro, Gabrielli Maurizio. Transformations of CLP modules // Theoretical computer science. — 1996. — Vol. 166, no. 1. — P. 101–146.

[10] Hoder Krystof, Bjørner Nikolaj. Generalized Property Directed Reachability. // SAT. — 2012. — Vol. 7317. — P. 157–171.

[11] ICE: a Robust Framework for Learning Invariants / Pranav Garg, Christof Löding, P Madhusudan, Daniel Neider // International Conference on Computer Aided Verification / Springer. — 2014. — P. 69–87.

[12] Jaffar Joxan, Lassez J-L. Constraint logic programming // Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages / ACM. — 1987. — P. 111–119.

[13] JayHorn: A framework for verifying Java programs / Temesghen Kahsai, Philipp Rümmer, Huascar Sanchez, Martin Schäf // International Conference on Computer Aided Verification / Springer. — 2016. — P. 352–358.

[14] Komuravelli Anvesh, Gurfinkel Arie, Chaki Sagar. SMT-based model checking for recursive programs // Formal Methods in System Design. — 2016. — Vol. 48, no. 3. — P. 175–205.

[15] Mordvinov Dmitry, Fedyukovich Grigory. Synchronizing Constrained Horn Clauses // LPAR, EPiC Series in Computing. EasyChair. — 2017.

[16] Mordvinov Dmitry, Fedyukovich Grigory. Verifying Safety of Functional Programs with Rosette/Unbound // arXiv preprint arXiv:1704.04558. — 2017.

[17] Program Verification via Iterated Specialization / Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti // Science of Computer Programming. — 2014. — Vol. 95. — P. 149–175.

[18] Relational verification through horn clause transformation / Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti // International Static Analysis Symposium / Springer. — 2016. — P. 147–169.

[19] Satisfiability Modulo Theories. / Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, Cesare Tinelli // Handbook of satisfiability. — 2009. — Vol. 185. — P. 825–885.

[20] The SeaHorn verification framework / Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli, Jorge A Navas // International Conference on Computer Aided Verification / Springer. — 2015. — P. 343–361.

[21] Sery Ondrej, Fedyukovich Grigory, Sharygina Natasha. Interpolation-Based Function Summaries in Bounded Model Checking. // Haifa verification conference / Springer. — 2011. — P. 160–175.

[22] Torlak Emina, Bodik Rastislav. A lightweight symbolic virtual machine for solver-aided host languages // ACM SIGPLAN Notices / ACM. — Vol. 49. — 2014. — P. 530–541.

[23] A verification toolkit for numerical transition systems / Hossein Hojjat, Filip Konečnỳ, Florent Garnier et al. // International Symposium on Formal Methods / Springer. — 2012. — P. 247–251.