

Санкт-Петербургский государственный университет

КОСТЮКОВ Юрий Олегович

Выпускная квалификационная работа

**Автоматический вывод регулярных
инвариантов программ с алгебраическими
типами данных**

Уровень образования: магистратура

Направление *09.04.04 «Программная инженерия»*

Основная образовательная программа *ВМ.5666.2019 «Программная инженерия»*

Научный руководитель:
д. т. н., профессор Д. В. Кознов

Рецензент:
ассистент-профессор университета штата Флорида, PhD Г. Г. Федюкович

Санкт-Петербург
2021

Saint Petersburg State University

Yurii Kostyukov

Master's Thesis

Automatic Inference of Regular Invariants of Programs with Algebraic Data Types

Education level: master

Speciality *09.04.04 “Software Engineering”*

Programme *BM.5666.2019 “Software Engineering”*

Scientific supervisor:
Sc.D, Prof. D. V. Koznov

Reviewer:
Assistant Professor at Florida State University, Ph.D. G. G. Fedyukovich

Saint Petersburg
2021

Оглавление

Введение	5
1. Постановка задачи	7
2. Обзор предметной области	8
2.1. Теория алгебраических типов данных	8
2.2. Представления инвариантов	8
2.3. Конечные модели и автоматы над деревьями	9
2.4. Алгебраические типы данных с ограничениями на размер термов	10
3. Основные понятия	11
3.1. Многосортная логика	11
3.2. Эрбрановская модель	11
3.3. Язык ограничений	12
3.4. Дизъюнкты Хорна с ограничениями	12
3.5. Безопасные индуктивные инварианты	13
3.6. Представления инвариантов	13
3.7. Конечные автоматы над деревьями	14
4. Регулярное представление инвариантов	15
5. Сравнительный анализ подходов к представлению инвариантов	17
5.1. Элементарное представление	17
5.2. Элементарное представление с ограничениями размера термов	20
5.3. Сравнение свойств представимости	22
6. Подход к выводу регулярных инвариантов	26
6.1. Трансляция в теорию неинтерпретированных функций	26
6.2. Построение автоматов над деревьями по конечным моделям	27
6.3. Эрбрановские модели без равенства	28
6.4. Эрбрановские модели с равенством	28
6.5. Устранение тестеров и селекторов	31
7. Сравнения и соотнесения	32
7.1. Инструмент Z3	32
7.2. Инструмент Eldarica	32
7.3. Другие инструменты с поддержкой АТД	32

8. Реализация и эксперименты	34
8.1. Пилотная реализация	34
8.2. Выбор инструментов для экспериментов	34
8.3. Набор данных	35
8.4. Результаты экспериментов	36
8.5. Выводы	39
Заключение	40
Список литературы	41

Введение

Для обеспечения безопасности программ кроме тестирования часто применяется верификация. Последняя подразумевает доказательство безопасности программы относительно некоторого свойства, выраженного на некотором языке спецификаций. Такие свойства часто закодированы в самой программе, например, внутри инструкции `assert`. Безопасность программы относительно свойства в таком случае будет означать, что программа никогда не завершит работу с ошибкой на инструкции `assert`. В качестве языков спецификаций используются логики Флойда-Хоара [21, 25], дизъюнкты Хорна с ограничениями (constrained Horn clauses) [4], аннотации программ с помощью ограничивающих типов (refinement types) [46] и т.д. Упомянутые подходы используют языки первого порядка (ЯПП) для описания свойств программ. Языки первого порядка являются простыми для восприятия, но при этом достаточно выразительными для описания содержательных свойств программ. Результатом верификации в случае безопасности программы относительно некоторого свойства должен быть проверяемый сертификат безопасности, например, индуктивный инвариант или резюме функции. Возможность автоматической проверки сертификата безопасности позволяет гарантировать корректность кода и отсутствие ошибок в самом верификаторе. Обычно эти сертификаты безопасности также основываются на языках первого порядка, что открывает путь к лучшему пониманию свойств программ для их авторов.

Выразительность языков первого порядка имеет свою цену: инструменты, автоматизирующие рассуждения о программах, например, инструменты автоматического доказательства теорем, решатели задач булевой выполнимости (Boolean satisfiability, SAT) и решатели задач выполнимости по модулю теорий (satisfiability modulo theory, SMT) требуют сложных алгоритмов для поддержки конкретных теорий и множество эвристик, что выливается в огромные затраты на обслуживание. Десятилетия уходят на то, чтобы сделать такие инструменты стабильными и производительными. Встроенные в них разрешающие процедуры чувствительны к конкретным фрагментам теорий первого порядка и нуждаются в постоянном улучшении. В частности, для теории алгебраических типов данных (АТД) предложены различные разрешающие процедуры [2, 43, 47, 53] и алгоритмы, осуществляющие интерполяцию Крейга [26, 31], реализованные в таких SMT-решателях, как Z3 [14], CVC4 [6] и PRINCESS [50]. Тем не менее, задача автоматической верификации программ с алгебраическими типами остаётся мало изученной и плохо решаемой на практике, из-за чего существующие верификаторы редко применяются вне университетов. Одна из возможных причин этого — плохая приспособленность языков первого порядка над АТД для задач верификации.

С одной стороны, языки первого порядка над АТД позволяют ограничивать термы

только на заранее заданную глубину, потому не являются достаточно выразительными в качестве языка спецификаций. С другой стороны, несмотря на простоту ЯПП, процедуры проверки выполнимости и интерполяционные процедуры для формул первого порядка являются асимптотически неэффективными. Таким образом, должны быть исследованы альтернативные языкам первого порядка представления инвариантов программ с АД, не ограничивающие глубину описываемых термов. Для таких представлений должны быть предъявлены эффективные процедуры вывода инвариантов в этих представлениях.

Данная работа направлена на исследование представлений инвариантов программ, отличных от языков первого порядка. В частности, в данной работе предлагается новый, основанный на *автоматах над деревьями*, класс представлений индуктивных инвариантов — класс *регулярных инвариантов*. Автоматы над деревьями позволяют определять свойства термов на заранее не ограниченной глубине. Также предлагается подход к автоматическому построению регулярных инвариантов при помощи инструментов поиска конечных моделей, таких как MACE4 [41], FINDER [51], PARADOX [10] и SVC4 [20]. Подход основывается на известном соответствии между конечными моделями и автоматами над деревьями и позволяет при помощи предобработки условий верификации и построения автоматов над деревьями по конечным моделям, получаемых из инструмента поиска конечных моделей, автоматически строить регулярные инварианты.

В рамках данной работы исследованы выразительные свойства трёх различных представлений инвариантов программ: двух существующих, основанных на языках первого порядка, и предлагаемого в данной работе регулярного представления. Сформулированы и доказаны теоремы, показывающие невыразительность обоих представлений первого порядка. Эти результаты являются теоретическим обоснованием того, почему такие решатели, как Z3 [33] и ELDARICA [27], не завершаются на безопасных программах. Таким образом, в данной работе проведён подробный теоретический анализ различных представлений инвариантов программ с АД и их выразительных свойств.

Также в рамках данной работы проведена реализация предлагаемого подхода по автоматическому выводу регулярных инвариантов. Проведено экспериментальное сравнение полученного инструмента и существующих решателей, поддерживающих теорию АД, Z3 [33] и ELDARICA [27]. Реализованный инструмент RINGEN вывел больше инвариантов программ с АД и был, в среднем, быстрее чем другие инструменты.

1. Постановка задачи

Целью данной работы является разработка подхода к автоматическому выводу инвариантов программ, использующих алгебраические типы данных, в рамках подходов по автоматической верификации функциональных программ. Для достижения этой цели были сформулированы следующие задачи.

- Предложить новое представление инвариантов для программ с алгебраическими типами данных.
- Провести сравнительный анализ предложенного представления с существующими.
- Предложить новый подход к выводу инвариантов программ с алгебраическими типами данных для предложенного представления инвариантов.
- Реализовать подход и провести эмпирическое сравнение с существующими подходами.

2. Обзор предметной области

2.1. Теория алгебраических типов данных

Алгебраические типы данных (АТД) чаще всего используются в функциональных языках программирования для определения рекурсивных неизменяемых структур данных, таких как списки и деревья (см. рисунок 1). Алгебраические типы данных могут строиться при помощи конструкторов, которыми они определяются (например, *Nil*, *Cons*, *Leaf*, *Node* на рисунке 1).

```
type list = Nil | Cons of int * list;;  
type tree = Leaf | Node of int * tree * tree;;
```

Рис. 1: Пример определения алгебраических типов данных на языке OCAML.

Алгебраические типы данных в рамках логической теории рассматриваются в связи с анализом функциональных программ [43]. Это делается для достижения модульности анализа: анализатор обрабатывает структуры языка программирования (например, транслирует операции с памятью в логические формулы), а сторонний *логический решатель* выполняет рассуждения в конкретных теориях (например, как связаны объекты в памяти, выраженные в формулах). Для такого модульного анализа были предложены SMT-решатели (satisfiability modulo theory) [5] и основанные на них Хорн-решатели [28].

Для решения задач анализа программ в логических решателях используются такие процедуры для теорий, как процедура устранения кванторов из формул, разрешающая процедуры для теории, процедура интерполяции формул. Процедура устранения кванторов из формул и процедура интерполяции позволяют обобщать свойства программ, например, относительно локальных переменных функций. Разрешающая процедура позволяет автоматически проверять корректность инвариантов программ, например, индуктивность инвариантов, то есть выполнение инварианта на всех шагах циклов или уровнях рекурсии. Было разработано множество алгоритмов устранения кванторов и разрешающих процедур для теории алгебраических типов данных [2, 43, 45, 47, 53]. В некоторых работах рассматривается также интерполяция Крейга для теории АТД [26, 31]. Эффективные версии этих процедур активно внедряются в различные SMT-решатели, такие как Z3 [14], CVC4 [6] и PRINCESS [50].

2.2. Представления инвариантов

Классически верификация программ проводится с выводом программных инвариантов, которые являются сертификатами безопасности программ [16]. Под инвариантом понимается свойство, сохраняющееся по ходу исполнения программы: на всех шагах цикла, на всех уровнях рекурсии и т.п.

С точки зрения логики (индуктивный) инвариант — это множество состояний программы, замкнутое относительно её отношения переходов. Для верификации используются *безопасные* индуктивные инварианты [42], то есть индуктивные инварианты, безопасные с точки зрения некоторого свойства безопасности. Так как свойство безопасности также описывает некоторое множество состояний, безопасным индуктивным инвариантом является только такой индуктивный инвариант, ни одно состояние которого не принадлежит множеству небезопасных состояний.

В данной работе инвариантами называются только безопасные индуктивные инварианты программ.

Так как чаще всего инварианты содержат бесконечное множество состояний программы, и они должны быть представлены в компьютерной памяти, вводят некоторое *представление инвариантов* [49], то есть способ конечным образом выразить бесконечное множество состояний.

Поскольку данная работа исследует программы с абстрактными типами данных, а *эрбрановские модели* являются начальными моделями теории АД [13], достаточно и в дальнейшем будут рассматриваться только такие модели. Эрбрановские модели в логике отличаются тем, что интерпретируют все константные и функциональные символы самими собой, и носители таких моделей состоят только из комбинаций таких символов. Существует ряд работ, рассматривающих различные представления эрбрановских моделей [18, 19, 23, 56]. В частности, известно множество эффективных алгоритмов для работы с автоматами над деревьями. Тем не менее, эти автоматы ограничены по своей выразительной силе, потому некоторые из их расширений широко изучались различными исследователями в области автоматического построения моделей [7]. Обзор вычислительных представлений эрбрановских моделей, их свойств, выразительной силы и эффективности необходимых для работы с ними процедур представлен в работах [39, 40].

2.3. Конечные модели и автоматы над деревьями

Автоматы над деревьями являются расширением классических строковых автоматов на языки деревьев (языки термов). Базовые результаты для автоматов над деревьями и их расширений представлены в работе [57].

Также исследуются различные расширения автоматов над деревьями, обладающие свойствами разрешимости и замкнутости базовых языковых операций (например, проверки на пустоту пересечения языков) [8, 9, 17, 22, 30, 36]. В контексте вывода инвариантов важно, что проверка *индуктивности* возможного инварианта (представленного конечным автоматом) разрешима, а проверка *существования* инварианта, представимого конечным автоматом — нет. Для проверки существования однако есть полурешающая процедура: если существует конечная модель, её можно найти пе-

ребором, но такой перебор не завершится, если система имеет только бесконечные модели.

Существует изоморфизм между представлениями множеств автоматами над деревьями и конечными моделями [57]. Поэтому построение автомата над деревьями для задачи эквивалентно поиску конечной модели для неё. Ряд инструментов, таких как MACE4 [41], FINDER [51], PARADOX [10] и CVC4 [20] могут быть использованы для поиска конечных моделей формул первого порядка в теории неинтерпретированных функций. Большинство из них реализуют классический DPLL (алгоритм Дэвиса-Патнема-Логемана-Лавленда)-подобный поиск с распространением равенств. CVC4 также использует анализ конфликтов для ускорения поиска. Данные инструменты применяются для задач верификации [37] и для построения бесконечных моделей [44]. Однако методы поиска конечных моделей до сих пор не использовались для вывода инвариантов программ, работающих с алгебраическими типами данных.

2.4. Алгебраические типы данных с ограничениями на размер термов

Для повышения выразительности языков первого порядка над АД было исследовано их расширение ограничениями на размер термов. Краткий обзор теории АД с ограничениями на размер термов дан в [60]. В частности, для этого класса предлагаются процедура устранения кванторов и разрешающая процедура.

Автоматический вывод индуктивных инвариантов в ЯПП с ограничениями размера термов реализован в Хорн-решателе ELDARICA [27]. Ограничения, определяемые формулами теории АД, в нём проверяются и обобщаются SMT-решателем PRINCESS [50], при помощи сведения в комбинацию теорий неинтерпретированных функций и линейной целочисленной арифметики [26]. Алгоритмы работы с формулами ЯПП с ограничениями на размер термов в этом решателе основываются на подходе, предложенном в работе [53].

3. Основные понятия

3.1. Многосортовая логика

Так как различные алгебраические типы данных в программе определяют различные, непересекающиеся сорта, теория алгебраических типов данных определяется в рамках многосортовой логики.

Многосортовая сигнатура первого порядка с равенством — это кортеж $\Sigma = \langle \Sigma_S, \Sigma_F, \Sigma_P \rangle$, где Σ_S — множество сортов, Σ_F — множество функциональных символов, Σ_P — множество предикатных символов, среди которых есть выделенный символ равенства $=_\sigma$ для каждого сорта σ . Каждый функциональный символ $f \in \Sigma_F$ имеет арность $\sigma_1 \times \dots \times \sigma_n \rightarrow \sigma$, где $\sigma_1, \dots, \sigma_n, \sigma \in \Sigma_S$, а каждый предикатный символ $p \in \Sigma_P$ имеет арность $\sigma_1 \times \dots \times \sigma_n$. Термы, атомы, формулы, замкнутые формулы и предложения языка первого порядка (ЯПП) определяются, как обычно.

Многосортовая структура (модель) $\mathcal{M}$ для сигнатуры Σ состоит из непустых носителей $|\mathcal{M}|_\sigma$ для каждого сорта $\sigma \in \Sigma_S$. Каждому функциональному символу f с арностью $\sigma_1 \times \dots \times \sigma_n \rightarrow \sigma$ сопоставлена интерпретация $M(f) : |\mathcal{M}|_{\sigma_1} \times \dots \times |\mathcal{M}|_{\sigma_n} \rightarrow |\mathcal{M}|_\sigma$, и каждому предикатному символу p с арностью $\sigma_1 \times \dots \times \sigma_n$ сопоставлена интерпретация $M(p) \subseteq |\mathcal{M}|_{\sigma_1} \times \dots \times |\mathcal{M}|_{\sigma_n}$. Для каждого замкнутого термина t с сортом σ интерпретация $\mathcal{M}[[t]] \in |\mathcal{M}|_\sigma$ определяется естественным образом. Структура называется конечной, если все её носители всех сортов конечны, в противном случае она называется бесконечной.

Выполнимость предложения φ в модели $\mathcal{M}$, обозначаемая $\mathcal{M} \models \varphi$, определяется как обычно. Употреблением $\varphi(x_1, \dots, x_n)$ вместо φ будет подчёркиваться, что все свободные переменные в φ находятся среди $\{x_1, \dots, x_n\}$. Далее, $\mathcal{M} \models \varphi(a_1, \dots, a_n)$ обозначает, что $\mathcal{M}$ выполняет φ на оценке, сопоставляющей свободным переменным элементы соответствующих носителей $a_1, \dots, a_n$ (переменные также связаны с сортами). Универсальное замыкание формулы $\varphi(x_1, \dots, x_n)$ обозначается $\forall \varphi$ и определяется как $\forall x_1 \dots \forall x_n. \varphi$. Если φ имеет свободные переменные, $\mathcal{M} \models \varphi$ означает $\mathcal{M} \models \forall \varphi$.

3.2. Эрбрановская модель

Эрбрановские модели являются начальными (интуитивно, наиболее естественными) моделями для теории АД, поэтому в данной работе без ограничения общности рассматриваются не все, а только эрбрановские модели утверждений.

Эрбрановский универсум для сорта σ — это множество всех замкнутых термов с сортом σ . Модель $\mathcal{H}$ называется *эрбрановской моделью* для сигнатуры Σ , если каждому сорту $\sigma \in \Sigma_S$ она ставит в соответствие эрбрановский универсум $|\mathcal{H}|_\sigma$ в качестве носителя, и интерпретирует каждый функциональный символ самим собой, т.е. $\mathcal{H}(f)(t_1, \dots, t_n) = f(\mathcal{H}[[t_1]], \dots, \mathcal{H}[[t_n]])$ для всех термов t_i с соответствующими сортами.

Таким образом, для каждой сигнатуры Σ существует семейство эрбрановских моделей с одинаковыми носителями и интерпретациями функциональных символов, но с различными интерпретациями предикатных символов. Очевидно, что каждая эрбрановская модель $\mathcal{H}$ интерпретирует каждый замкнутый терм t самим собой, то есть $\mathcal{H}[\![t]\!] = t$.

3.3. Язык ограничений

Язык ограничений фиксирует способ представления свойств программы в виде логических формул. Для осуществления анализа программ её свойства кодируются анализатором в языке ограничений и передаются в сторонний логический решатель.

Алгебраический тип данных (АТД) — это кортеж $\langle C, \sigma \rangle$, где σ — это сорт, а C — это множество функциональных символов (называемых конструкторами), таких что каждый $f \in C$ имеет арность $\sigma_1 \times \dots \times \sigma_n \rightarrow \sigma$ для некоторых сортов $\sigma_1, \dots, \sigma_n$.

Пусть дан набор АТД $\langle C_1, \sigma_1 \rangle, \dots, \langle C_n, \sigma_n \rangle$, такой что $\sigma_i \neq \sigma_j$ и $C_i \cap C_j = \emptyset$ при $i \neq j$. В связи с фокусом данной работы далее будет рассматриваться только сигнатура теории алгебраических типов данных¹ $\Sigma = \langle \Sigma_S, \Sigma_F, \Sigma_P \rangle$, где $\Sigma_S = \{\sigma_1, \dots, \sigma_n\}$, $\Sigma_F = C_1 \cup \dots \cup C_n$ и $\Sigma_P = \{=_{\sigma_1}, \dots, =_{\sigma_n}\}$. Для упрощения обозначений в символах равенства будут опускаться индексы сортов. Язык первого порядка, определённый над сигнатурой Σ , будет называться *языком ограничений* $\mathcal{L}$.

Поскольку Σ не имеет предикатных символов, отличных от символов равенства (которые имеют фиксированные интерпретации внутри каждой структуры), существует единственная эрбрановская модель $\mathcal{H}$ для Σ . Тем самым, предложение (формула) φ языка ограничений будет называться *выполнимой по модулю теории АТД* $\langle C_1, \sigma_1 \rangle, \dots, \langle C_n, \sigma_n \rangle$, если $\mathcal{H} \models \varphi$.

3.4. Дизъюнкты Хорна с ограничениями

В отличие от формул языка ограничений, которые описывают свойства одного отдельного состояния, дизъюнкты Хорна с ограничениями позволяют описывать все достижимые состояния программы. Тем самым анализ программ может быть превращён в синтаксическую процедуру трансляции программы в систему дизъюнктов Хорна с ограничениями [28], которая уже передаётся в логический решатель дизъюнктов Хорна.

Пусть $\mathcal{R} = \{P_1, \dots, P_n\}$ — конечное множество отличных от равенства предикатных символов с сортами из Σ . Такие символы будут называться *неинтерпретированными*. Формула C над сигнатурой $\Sigma \cup \mathcal{R}$ называется *дизъюнктом Хорна с ограничениями*.

¹Для простоты изложения в сигнатуре опущены селекторы и тестеры, поскольку они не влияют на выразительность языка ограничений.

ниями, если это формула следующего вида.

$$\varphi \wedge R_1(\bar{t}_1) \wedge \dots \wedge R_m(\bar{t}_m) \rightarrow H$$

Здесь φ — *ограничение* (формула языка ограничений без кванторов), $R_i \in \mathcal{R}$, $\bar{t}_i$ — кортеж термов. H называется *головой* дизъюнкта и может быть либо $\perp$ (тогда дизъюнкт называется *запросом*), либо атомарной формулой $R(\bar{t})$ (тогда дизъюнкт называется *определением для R*), $R \in \mathcal{R}$ и $\bar{t}$ — это кортеж термов. Посылка импликации $\varphi \wedge R_1(\bar{t}_1) \wedge \dots \wedge R_m(\bar{t}_m)$ называется *телом C* .

Системой дизъюнктов Хорна $\mathcal{S}$ называется конечное множество дизъюнктов Хорна с ограничениями.

3.5. Безопасные индуктивные инварианты

Безопасные индуктивные инварианты программ являются сертификатами их безопасности. При наличии такого сертификата можно автоматически проверить, что программа удовлетворяет сертификату относительного некоторого свойства безопасности, тем самым подтвердив безопасность [42].

Пусть $\bar{X} = \langle X_1, \dots, X_n \rangle$ — кортеж отношений, такой что если P_i имеет сорт $\sigma_1 \times \dots \times \sigma_m$, то $X_i \subseteq |\mathcal{H}|_{\sigma_1} \times \dots \times |\mathcal{H}|_{\sigma_m}$. Для упрощения обозначений, расширение $\mathcal{H}\{P_1 \mapsto X_1, \dots, P_n \mapsto X_n\}$ будет записываться как $\langle \mathcal{H}, X_1, \dots, X_n \rangle$ или просто $\langle \mathcal{H}, \bar{X} \rangle$.

Система дизъюнктов Хорна $\mathcal{S}$ называется *выполнимой по модулю теории АТД*, если существует кортеж отношений $\bar{X}$, такой что $\langle \mathcal{H}, \bar{X} \rangle \models C$ для всех $C \in \mathcal{S}$. В таком случае кортеж $\bar{X}$ называется (*безопасным индуктивным*) *инвариантом* системы $\mathcal{S}$.

Хорошо известно, что дизъюнкты Хорна с ограничениями являются фрагментом логики первого порядка, соответствующим множеству программных логик, таких как логика Флойда-Хоара для императивных программ и ограничивающие типы для функциональных программ. Далее для каждой рекурсивной программы над АТД будет рассматриваться система дизъюнктов, из выполнимости которой следует безопасность исходной программы. Тем самым, в дальнейшем повествовании программы будут идентифицироваться с их условиями безопасности, выраженными в виде дизъюнктов Хорна.

3.6. Представления инвариантов

Представление инварианта — это функция $\mathcal{C}$, отображающая каждый кортеж сортов $\langle \sigma_1, \dots, \sigma_n \rangle \in \Sigma_S^n$ для каждого $n \in \mathbb{N}$ в некоторый класс языков $\mathcal{C}(\sigma_1, \dots, \sigma_n) \subseteq 2^{|\mathcal{H}|_{\sigma_1} \times \dots \times |\mathcal{H}|_{\sigma_n}}$. Говорится, что отношение $X \subseteq |\mathcal{H}|_{\sigma_1} \times \dots \times |\mathcal{H}|_{\sigma_n}$ *представимо* в классе $\mathcal{C}$, если $X \in \mathcal{C}(\sigma_1, \dots, \sigma_n)$. Расширение эрбрановской модели $\langle \mathcal{H}, \bar{X} \rangle$ *представимо* в $\mathcal{C}$ (или *$\mathcal{C}$ -представимо*), если каждое отношение $X \in \bar{X}$ представимо в $\mathcal{C}$.

3.7. Конечные автоматы над деревьями

В этом разделе вводятся детерминированные конечные *автоматы над деревьями*. Определения заимствованы из работы [57].

Пусть $\Sigma = \langle \cdot, \Sigma_F, \cdot \rangle$ — некоторая фиксированная многосортная сигнатура.

Определение 1. *Детерминированный конечный n -автомат над деревьями (ДКАД) над Σ_F — это кортеж $(S, \Sigma_F, S_F, \Delta)$, где S — (конечное) множество состояний, $S_F \subseteq S^n$ — множество конечных состояний, Δ — отношение перехода с правилами следующего вида.*

$$f(s_1, \dots, s_m) \rightarrow s$$

Здесь $f \in \Sigma_F$, $ar(f) = m$ и $s, s_1, \dots, s_m \in S$ и в Δ нет правил с совпадающей левой частью.

Определение 2. Кортеж замкнутых термов $\langle t_1, \dots, t_n \rangle$ *принимается* (допускается) n -автоматом $A = (S, \Sigma_F, S_F, \Delta)$, если $\langle A[t_1], \dots, A[t_n] \rangle \in S_F$, где

$$A[f(t_1, \dots, t_m)] \stackrel{\text{def}}{=} \begin{cases} s, & \text{если } (f(A[t_1], \dots, A[t_m]) \rightarrow s) \in \Delta, \\ \text{не определено,} & \text{иначе.} \end{cases}$$

Пример 1. Пусть $\Sigma = \langle Prop, \{(_ \wedge _), (_ \vee _), (_ \rightarrow _), \top, \perp\}, \emptyset \rangle$ сигнатура логики высказываний. Рассмотрим автомат $A = (\{q_0, q_1\}, \Sigma_F, \{q_1\}, \Delta)$ со следующим набором отношений перехода Δ .

$$\begin{array}{ll} q_0 \vee q_0 \mapsto q_0 & \\ q_0 \vee q_1 \mapsto q_1 & \perp \mapsto q_0 \\ q_1 \vee q_0 \mapsto q_1 & \top \mapsto q_1 \\ q_1 \vee q_1 \mapsto q_1 & q_1 \rightarrow q_0 \mapsto q_0 \\ q_1 \wedge q_1 \mapsto q_1 & q_1 \rightarrow q_1 \mapsto q_1 \\ q_1 \wedge q_0 \mapsto q_0 & q_0 \rightarrow q_0 \mapsto q_1 \\ q_0 \wedge q_1 \mapsto q_0 & q_0 \rightarrow q_1 \mapsto q_1 \\ q_0 \wedge q_0 \mapsto q_0 & \end{array}$$

A допускает только истинные пропозициональные формулы без переменных.

4. Регулярное представление инвариантов

В данном разделе предлагается новое представление инвариантов программ с алгебраическими типами, основанное на автоматах над деревьями термов, а не на языках первого порядка, как это делается традиционно. Такой подход позволяет выражать свойства алгебраических термов на произвольном уровне глубины. Благодаря важным свойствам — замкнутости относительно булевых операций над языками, разрешимости проверки на пустоту и проверки вхождения слова в язык — автоматы над деревьями являются эффективной заменой языков первого порядка для представления инвариантов.

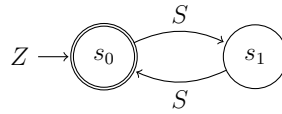
Пусть $\mathcal{H}$ — эрбрановская модель для сигнатуры $\langle \cdot, \Sigma_F, \cdot \rangle$. Будем говорить, что n -автомат A над Σ_F представляет отношение $X \subseteq |\mathcal{H}|_{\sigma_1} \times \dots \times |\mathcal{H}|_{\sigma_n}$, если $X = \{ \langle a_1, \dots, a_n \rangle \mid \langle a_1, \dots, a_n \rangle \text{ принимается } A, a_i \in |\mathcal{H}|_{\sigma_i} \}$. Если есть ДКАД, представляющий отношение X , то X называется *регулярным* отношением. Класс регулярных отношений будет обозначаться REG.

Пусть $\mathcal{S}$ — система дизъюнктов Хорна. Если $\bar{X} = \langle X_1, \dots, X_n \rangle$, где каждый X_i регулярен, и $\langle \mathcal{H}, \bar{X} \rangle \models C$ для всех $C \in \mathcal{S}$, тогда $\langle \mathcal{H}, \bar{X} \rangle$ называется *регулярной моделью* или *регулярным инвариантом* $\mathcal{S}$.

Пример 2 (Even). Рассмотрим следующую систему дизъюнктов Хорна над алгебраическим типом $Nat ::= Z \mid S \ Nat$.

$$\begin{aligned} even(x) &\leftarrow x = Z \\ even(x) &\leftarrow x = S(S(y)) \wedge even(y) \\ \perp &\leftarrow even(x) \wedge even(y) \wedge y = S(x) \end{aligned}$$

Она имеет регулярную модель $\langle \mathcal{H}, X \rangle$, порождённую 1-ДКАД $(\{s_0, s_1, s_2\}, \Sigma_F, \{s_0\}, \Delta)$, где Δ состоит из правил перехода следующего вида:



Имеем $\langle \mathcal{H}, X \rangle(even) = X = \{Z, S(S(Z)), S(S(S(S(Z))))\}, \dots\} = \{S^{2n}(Z) \mid n \geq 0\}$.

Важно заметить, что в общем случае, система дизъюнктов может иметь более одного инварианта. Однако большинство примеров в данной работе подобраны так, чтобы иметь уникальный инвариант. Например, единственным инвариантом системы из примера 2 является множество $E = \{Z, S(S(Z)), \dots\}$. Это можно показать от противного: если расширить это множество некоторым нечетным числом $E \cup \{S^{2n+1}(Z)\} \subseteq E'$, то будет нарушено условие запроса при $x = S^{2n}(Z)$ и $y = S^{2n+1}(Z)$. Таким образом, E — единственный безопасный индуктивный инвариант.

Пример 3 (*IncDec*). Рассмотрим следующую систему дизъюнктов с несколькими различными инвариантами.

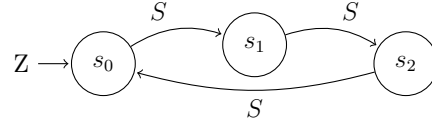
$$\begin{aligned}
inc(x, y) &\leftarrow x = Z \wedge y = S(Z) \\
inc(x, y) &\leftarrow x = S(x') \wedge y = S(y') \wedge inc(x', y') \\
dec(x, y) &\leftarrow x = S(Z) \wedge y = Z \\
dec(x, y) &\leftarrow x = S(x') \wedge y = S(y') \wedge dec(x', y') \\
\perp &\leftarrow inc(x, y) \wedge dec(x, y)
\end{aligned}$$

Эта система имеет очевидный инвариант, представимый в языке ограничений следующим образом:

$$inc(x, y) \equiv (y = S(x)), dec(x, y) \equiv (x = S(y)).$$

Этот инвариант является наиболее сильным из возможных, т.к. выражает денотационную семантику *inc* и *dec* соответственно. Эти отношения нерегулярны, то есть не существует 2-автоматов представляющих эти отношения [57].

Однако эта система дизъюнктов имеет другой, менее очевидный, регулярный инвариант, порождённый двумя 2-ДКАД $(\{s_0, s_1, s_2, s_3\}, \Sigma_F, Q_*, \Delta)$ с конечными состояниями соответственно $Q_{inc} = \{\langle s_0, s_1 \rangle, \langle s_1, s_2 \rangle, \langle s_2, s_0 \rangle\}$, $Q_{dec} = \{\langle s_1, s_0 \rangle, \langle s_2, s_1 \rangle, \langle s_0, s_2 \rangle\}$ и с правилами перехода, имеющими следующий вид:



Автомат для *inc* проверяет, что $(x \bmod 3, y \bmod 3) \in \{(0, 1), (1, 2), (2, 0)\}$, а автомат для *dec* проверяет, что $(x \bmod 3, y \bmod 3) \in \{(1, 0), (2, 1), (0, 2)\}$. Эти отношения аппроксимируют сверху денотационную семантику *inc* и *dec* и при этом доказывают невыполнимость формулы $inc(x, y) \wedge dec(x, y)$.

5. Сравнительный анализ подходов к представлению инвариантов

В данном разделе проводится анализ выразительных свойств различных представлений инвариантов путём сравнения, какие инварианты выражаются при помощи каких представлений. Рассматриваются регулярное представление и два представления, основанные на языках первого порядка, которые используются в современных инструментах вывода инвариантов — *элементарное представление*, применённое в инструменте Z3 [33], и *элементарным представлением с ограничениями размера термов*, используемое в инструменте ELDARICA [27].

Несмотря на то, что автоматы над деревьями эквивалентны монадической логике второго порядка над деревьями [57], которая несравнима с языками первого порядка, их соотношение требует детального изучения, которое и проведено в этом разделе.

Таким образом, цель данного раздела — предложить средство опровержения представимости отношений в ЯПП. С практической точки зрения изучение краевых случаев представимости полезно тем, что позволяет понять, почему инструменты вывода инвариантов, такие как Z3 или ELDARICA, расходятся на безопасных системах с непредставимыми инвариантами. Если инвариант безопасной системы непредставим в классе, используемом в конкретном инструменте, то этот инструмент будет бесконечно строить инвариант и не завершит свою работу. Для этой цели в данном разделе предложены *леммы о накачке* для двух элементарных представлений.

Первые леммы о накачке возникли в теории формальных языков в связи с конечными автоматами и контекстно-свободными грамматиками. В общем виде лемма о накачке должна показывать, что для всех языков в некотором классе (например, регулярных или контекстно-свободных языков) любое достаточно большое слово может быть «накачено», т.е. некоторые его части могут быть неограниченно увеличены, и накачанное слово при этом останется в языке. Леммы о накачке полезны для доказательства непредставимости инвариантов к некоторому классу: предположив, что инвариант принадлежит некоторому классу, можно применить лемму о накачке для этого класса и получить некоторый «накачанный» элемент, который не может лежать в инварианте. Такое противоречие доказывает, что инвариант непредставим в данном классе.

5.1. Элементарное представление

Отношение $X \subseteq |\mathcal{M}|_{\sigma_1} \times \dots \times |\mathcal{M}|_{\sigma_n}$ называется *представимым языком первого порядка* или *элементарным*, если существует формула $\varphi(x_1, \dots, x_n)$ в языке ограниченный, такая что $(a_1, \dots, a_n) \in X$ тогда и только тогда, когда $\mathcal{H} \models \varphi(a_1, \dots, a_n)$. Класс элементарно-представимых отношений будем обозначать ELEM.

Рассмотрим систему дизъюнктов из примера 3.

$$\begin{aligned}
x &= Z \wedge y = S(Z) \rightarrow inc(x, y) \\
x &= S(x') \wedge y = S(y') \wedge inc(x', y') \rightarrow inc(x, y) \\
x &= S(Z) \wedge y = Z \rightarrow dec(x, y) \\
x &= S(x') \wedge y = S(y') \wedge dec(x', y') \rightarrow dec(x, y) \\
inc(x, y) \wedge dec(x, y) &\rightarrow \perp
\end{aligned}$$

Как уже было замечено, её денотационная семантика представляется элементарно следующим образом:

$$inc(x, y) \equiv (y = S(x)), dec(x, y) \equiv (x = S(y)).$$

Однако не у всех систем есть элементарно-представимый инвариант. Чтобы доказывать

5.1.1. Лемма о накачке для элементарного представления

В этом разделе предложено средство для опровержения элементарной представимости — лемма о накачке для языков первого порядка. Различные применения леммы будут продемонстрированы на системе *Even* (см. пример 2).

Высота замкнутого терма определяется индуктивно следующим образом:

$$\begin{aligned}
Height(c) &\stackrel{\text{def}}{=} 1, \\
Height(c(t_1, \dots, t_n)) &\stackrel{\text{def}}{=} 1 + \max_{i=1}^n (Height(t_i)).
\end{aligned}$$

Для каждого АТД $\langle \sigma, C \rangle$ и каждого конструктора $f \in C$, имеющего арность $\sigma_1 \times \dots \times \sigma_n \rightarrow \sigma$ для некоторых сортов $\sigma_1, \dots, \sigma_n$, введём *селекторы* $g_i \in S$ с арностью $\sigma \rightarrow \sigma_i$ для каждого $i \leq n$, со стандартной семантикой: $g_i(f(t_1, \dots, t_n)) \stackrel{\text{def}}{=} t_i$.

Будем называть *путём* следующую последовательность селекторов (возможно пустую): для $S_n : \sigma_n \rightarrow \sigma_{n-1}, \dots, S_1 : \sigma_1 \rightarrow \sigma_0$ путь определяется как $s \stackrel{\text{def}}{=} S_1 \dots S_n$. Для термов t сорта σ_n пусть $s(t) \stackrel{\text{def}}{=} S_1(\dots(S_n(t))\dots)$. Для замкнутых термов g переопределим $s(g)$ как вычисленный подтерм g в s . В дальнейшем пути будем обозначать прописными буквами p, q, r, s .

Мы говорим, что два пути p и q *перекрываются*, если один из них является суффиксом другого. Для попарно неперекрывающихся путей $p_1, \dots, p_n$ с помощью записи $t[p_1 \leftarrow u_1, \dots, p_n \leftarrow u_n]$ будем понимать терм, полученный одновременной заменой подтермов $p_i(t)$ на термы u_i в t . Для конечной последовательности попарно различных путей $P = (p_1, \dots, p_n)$ и некоторого множества термов $U = (u_1, \dots, u_n)$ мы переопределяем обозначения и пишем $t[P \leftarrow U]$ вместо $t[p_1 \leftarrow u_1, \dots, p_n \leftarrow u_n]$, а также $t[P \leftarrow t]$ вместо $t[p_1 \leftarrow t, \dots, p_n \leftarrow t]$.

Теперь определим множество путей, которое будет накачиваться.

Определение 3. Для всех сортов σ терм t является *листовым термом* сорта σ , если это конструктор без параметров, или $t = c(t_1, \dots, t_n)$, где все t_i являются листовыми

термами, а t не содержит никаких собственных подтермов сорта σ . Для замкнутого терма g и сорта σ мы определяем $leaves_\sigma(g) \stackrel{\text{def}}{=} \{p \mid p(g) \text{ — листовой терм сорта } \sigma\}$.

Лемма 1 (Лемма о накачке для класса ЕЛЕМ). *Для любого элементарного языка n -кортежей $\mathcal{L}$ существует константа $K > 0$, такая что для каждого кортежа термов $\langle g_1, \dots, g_n \rangle \in \mathbf{L}$, для любого i , такого что $\text{Height}(g_i) > K$, для всех бесконечных сортов $\sigma \in \Sigma_S$ и для всех путей p в терме g с глубиной большей K , существует набор конечных последовательностей путей P_j , где $p \in P_i$ и для всех $p_1, p_2 \in \bigcup_j P_j$ верно что $p_1(g) = p_2(g)$, и найдётся константа $N \geq 0$, такие что для любого терма t сорта σ с $\text{Height}(t) > N$, имеет место следующее:*

$$\langle g_1[P_1 \leftarrow t], \dots, g_i[P_i \leftarrow t], \dots, g_n[P_n \leftarrow t] \rangle \in \mathbf{L}.$$

Доказательство. Полное доказательство приведено в [34, 35]. □

Интуитивно, лемма 1 утверждает, что для достаточно больших кортежей термов можно взять любой из самых глубоких подтермов, заменить их на *произвольный* терм t и всё ещё получить кортеж термов из данного языка. Лемма формализует интуицию о том, что язык ограничений над теорией АТД может описывать только равенства и неравенства между подтермами ограниченной глубины: если пойти достаточно глубоко и заменить листовые термы произвольными термами, то начальный и результирующий термы будут *неотличимы* с помощью формул языка первого порядка.

Теперь покажем, как использовать лемму о накачке для доказательства непредставимости инвариантов.

Покажем, что инвариант из примера *Even* не является элементарным.

Свойство 1. *Инвариант системы из примера Even не может быть представлен элементарно.*

Доказательство. Покажем, что $Even \notin \text{ЕЛЕМ}$. Предположим, что существует элементарно-представимый инвариант $\mathbf{L}$. Возьмём постоянную $K > 0$ из леммы о накачке 1. Пусть $g \equiv S^{2K}(Z) \in \mathbf{L}, \sigma = \text{Nat}, p = S^{2K}$. Далее, $\bigcup_j leaves_\sigma(g_j) = leaves_\sigma(g) = \{p\}$, поэтому $P = \{p\}$. Тогда по лемме о накачке найдётся $N \geq 0$; возьмём $t \equiv S^{2N+1}(Z)$. По лемме $g[P \leftarrow t] \equiv S^{2K}(S^{2N+1}(Z)) \in \mathbf{L}$. Но тогда $\mathbf{L}$ не выполняет последний дизъюнкт системы, поэтому $\mathbf{L}$ не является инвариантом, а это — противоречие. □

Таким образом, элементарное представление инвариантов не позволяет выражать некоторые инварианты программ с АТД. Существует представление, повышающее выразительность языка ограничений АТД путём расширения языка первого порядка некоторыми дополнительными символами, которое сохраняет разрешимость проверки выполнимости формул.

5.2. Элементарное представление с ограничениями размера термов

Представляется естественной идея увеличить выразительность элементарных представлений путём расширения языка первого порядка возможностью указывать *размеры термов*. Вывод инвариантов, представимых в этом классе, автоматизирован в инструменте ELDARICA [27].

Язык класса SIZEELEM может быть получен из языка ELEM путем добавления в сигнатуру сорта Int , операций из арифметики Пресбургера и функциональных символов $size_\sigma$ с арностью $\sigma \rightarrow Int$, которые считают конструкторы в термах сорта σ . Для краткости мы будем опускать σ в символах $size$.

Выполнимость формул с ограничениями размера термов проверяется в структуре $\mathcal{H}_{size}$, полученной путём соединения стандартной модели арифметики Пресбургера с эрбрановской моделью $\mathcal{H}$ и описанной выше интерпретации функциональных символов $size_\sigma$. Например, размер терма $t \equiv cons(Z, cons(S(Z), nil))$ с сортом $List$, определяемым как $List := nil : List \mid cons : Nat \times List \rightarrow List$, в объединённой структуре вычисляется следующим образом: $\mathcal{H}_{size} \llbracket size(t) \rrbracket = 6$.

Заимствуя обозначения [26], обозначим $\mathbb{T}_\sigma^k = \{t \text{ имеет сорт } \sigma \mid size(t) = k\}$. Для каждого АТД-сорта σ мы определяем множество размеров термов $\mathbb{S}_\sigma = \{size(t) \mid t \in |\mathcal{H}|_\sigma\}$. *Линейное множество* — это множество вида $\{\mathbf{v} + \sum_{i=1}^n k_i \mathbf{v}_i \mid k_i \in \mathbb{N}_0\}$, где все $\mathbf{v}, \mathbf{v}_i$ — векторы над $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

Определение 4. АТД-сорт σ называется *расширяющимся*, если для каждого натурального числа n существует граница $b(\sigma, n) \geq 0$ такая, что для каждого $b' \geq b(\sigma, n)$, если $\mathbb{T}_\sigma^{b'} \neq \emptyset$, то $|\mathbb{T}_\sigma^{b'}| \geq n$. Сигнатура АТД называется расширяющейся, если все её сорта расширяющиеся.

Хотя ограничения на размер термов существенно расширяют выразительные возможности языков первого порядка в качестве представления инвариантов, они не позволяют преодолеть их внутренние ограничения, одним из которых является возможность фиксировать термы только на ограниченную глубину. Чтобы продемонстрировать эту проблему, рассмотрим небольшую модификацию системы из примера 2.

Пример 4 (*EvenLeft*). Возьмём алгебраический тип бинарного дерева $Tree := leaf : Tree \mid node(Left : Tree, Right : Tree)$ и программу *EvenLeft*, которая проверяет, чётное ли количество узлов в самой левой ветви дерева, и определяется следующей системой дизъюнктов.

$$\begin{aligned} x = leaf &\rightarrow EvenLeft(x) \\ x = node(node(x', y), z) \wedge EvenLeft(x') &\rightarrow EvenLeft(x) \\ EvenLeft(x) \wedge EvenLeft(node(x, y)) &\rightarrow \perp \end{aligned}$$

Заметим, что языки первого порядка с ограничениями размера термов допускают устранение кванторов [60], поэтому они могут накладывать ограничения только на некоторые подтермы *конечной* глубины. Кроме того, они учитывают все конструкторы одновременно — мы не можем указать в формуле, что считать нужно только конструкторы самой левой ветви, как в примере *EvenLeft* выше, так что система из этого примера не имеет инварианта, представимого в классе SIZEELEM. Это можно показать более формально при помощи леммы о накачке для этого представления.

Лемма 2 (Лемма о накачке для класса SIZEELEM). *Пусть есть некоторая расширяющая АТД-сигнатура и $\mathbf{L}$ — элементарный язык n -кортежей с ограничениями на размер термов. Существует константа $K > 0$, такая что: для каждого кортежа термов $\langle g_1, \dots, g_n \rangle \in \mathbf{L}$, для любого i , такого что $\text{Height}(g_i) > K$, для всех бесконечных сортов $\sigma \in \Sigma_S$ и для всех путей p в терме g с глубиной большей K , существует бесконечное линейное множество $T \subseteq \mathbb{S}_\sigma$, такое что для любых термов t сорта σ с размером $\text{size}(t) \in T$, существует набор последовательностей путей P_j , ни один путь из которых не является суффиксом пути p и набор последовательностей термов U_j , такие что*

$$\langle g_1[p_1 \leftarrow U_1], \dots, g_i[p \leftarrow t, P_i \leftarrow U_i], \dots, g_n[P_n \leftarrow U_n] \rangle \in \mathbf{L}.$$

Доказательство. Полное доказательство приведено в [34, 35]. □

Интуитивно, имея язык из класса SIZEELEM, достаточно большой терм g из него и достаточно большой путь p , можно заменить $p(g)$ произвольным термом t (ограниченным только размером, который должен находиться в некотором линейном бесконечном множестве T), и вновь получить терм из языка. Это означает, что для языков из класса SIZEELEM существуют неограничиваемые подтермы, которые неотличимы в этом представлении. Продемонстрируем лемму о накачке на примере, приведённом выше.

Свойство 2. *Инвариант системы из примера *EvenLeft* не может быть представлен элементарной формулой с ограничениями на размер термов.*

Доказательство. Используя лемму о накачке, мы можем доказать, что инвариант *EvenLeft* не лежит в классе SIZEELEM. Во-первых, очевидно, что сорт *Tree* расширяющийся. Предположим, что *EvenLeft* находится в классе SIZEELEM и имеет инвариант $\mathbf{L}$. Возьмём $K > 0$ из леммы. Пусть $g \in \mathbf{L}$ будет полным двоичным деревом высоты $2K$, $\sigma = \text{Tree}, p = \text{Left}^{2K}$. Возьмём бесконечное линейное множество T из леммы. Мы можем найти некоторый $n \in T, n > 2$ и $t = \text{node}(\text{leaf}, t')$ для некоторого t' , такие что $\text{size}(t) = n$. По лемме существует последовательность путей P и последовательность термов U , и ни один из элементов в P не является суффиксом p . По лемме также должно быть верно, что $g[p \leftarrow t, P \leftarrow U] \in \mathbf{L}$, поэтому крайний левый путь

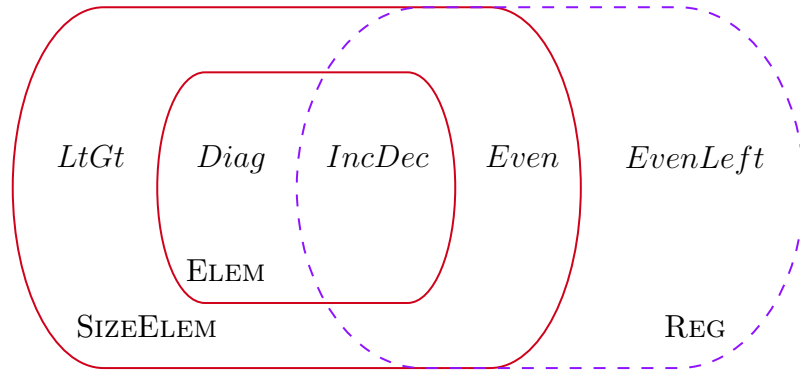


Рис. 2: Сопоставление трёх классов представлений инвариантов программ над АД.

в дереве должен иметь чётную длину. Однако по крайнему левому пути $p = Left^{2K}$ лежит терм $node(leaf, t')$, поэтому путь до крайнего левого листа дерева имеет длину $2K - 1 + 2 = 2 + 1$, нечётное число. Противоречие. $\square$

Теоретически ограничения размера увеличивают выразительную силу языка ограничений. Однако на практике может случиться так, что автоматический вывод инвариантов в классе SIZEELEM будет значительно медленнее из-за проверки выполнимости в расширенном языке и обобщения ограничений размера. Анализ эффективности инструмента ELDARICA в сравнении с другими инструментами приведён в разделе 8.

5.3. Сравнение свойств представимости

В данном разделе приведены взаимные соотношения классов ELEM, REG и SIZEELEM. Результаты раздела представлены на рисунке 2. Области на рисунке обозначают классы представления: самый маленький соответствует классу ELEM, большой слева соответствует классу SIZEELEM, а пунктирная область справа обозначает класс REG. Имена, выделенные курсивом, обозначают системы дизъюнктов. Если система находится внутри некоторого региона, значит она имеет инвариант, представимый в соответствующем классе.

Свойство 3. *Существуют инварианты, представимые как элементарно, так и регулярно.*

Доказательство. Покажем, что $ELEM \cap REG \neq \emptyset$. Система $IncDec \in ELEM \subseteq SIZEELEM$ (см. раздел 5.1), а также $IncDec \in REG$ (см. пример 3), поэтому пересечение этих классов не пустое. $\square$

Используя лемму о накачке для класса ELEM, мы показали в свойстве 1, что *Even* не имеет элементарного инварианта, поэтому имеем следующий факт.

Свойство 4. *Существуют инварианты, представимые регулярно, но не элементарно.*

Доказательство. Тот факт, что $\text{REG} \setminus \text{ELEM} \neq \emptyset$ следует из примера 2 и свойства 1. $\square$

Однако ограничения на размер термов позволяют представить инвариант из примера *Even*, как описано в следующем свойстве.

Свойство 5. *Инвариант программы из примера Even представим элементарной формулой с ограничениями размера термов.*

Доказательство. Покажем, что $\text{Even} \in \text{SIZEELEM}$. *Even* имеет безопасный индуктивный инвариант, определяемый формулой

$$\text{even}(x) \Leftrightarrow \exists n. \text{size}(x) = 1 + n + n.$$

$\square$

Небольшая модификация приведённого выше примера *Even* даёт систему *EvenLeft* из примера 4, у которой нет инварианта, представимого в классе SIZEELEM , однако есть регулярно-представимый инвариант.

Свойство 6. *Инвариант программы из примера EvenLeft представим регулярно.*

Доказательство. Покажем, что $\text{EvenLeft} \in \text{REG}$. Инвариант этой системы выражается автоматом $(\{s_0, s_1, s_2\}, \Sigma_F, \{s_0\}, \Delta)$ с правилами перехода Δ , описываемыми следующим образом.

$$\begin{aligned} \text{leaf} &\mapsto s_0 \\ \text{node}(s_0, s_0) &\mapsto s_1 \\ \text{node}(s_0, s_1) &\mapsto s_1 \\ \text{node}(s_1, s_0) &\mapsto s_0 \\ \text{node}(s_1, s_1) &\mapsto s_0 \end{aligned}$$

$\square$

Свойство 7. *Существуют инварианты, представимые регулярно, но не представимые элементарными формулами с ограничениями размера термов.*

Доказательство. Тот факт, что $\text{REG} \setminus \text{SIZEELEM} \neq \emptyset$, следует из свойства 6 и свойства 2. $\square$

Таким образом, во многих случаях регулярное представление более выразительно, чем оба элементарных. Однако существуют программы с элементарными, но не регулярными инвариантами.

Пример 5 (*Diag*). Рассмотрим систему дизъюнктов над целыми числами Пеано $Nat := Z : Nat \mid S : Nat \rightarrow Nat$ с предикатом $diseq$, рекурсивно проверяющим неравенство двух термов.

$$\begin{aligned} eq(x, y) &\leftarrow x = y \\ diseq(x, y) &\leftarrow x = S(x') \wedge y = Z \\ diseq(x, y) &\leftarrow y = S(y') \wedge x = Z \\ diseq(x, y) &\leftarrow x = S(x') \wedge y = S(y') \wedge diseq(x', y') \\ \perp &\leftarrow eq(x, y) \wedge diseq(x, y) \end{aligned}$$

Несмотря на то, что регулярные языки замкнуты относительно пересечения, объединения и дополнения, автоматы над деревьями не позволяют представлять отношения неравенства, что не позволяет регулярным представлениям «поглотить» элементарные. Основная причина этого заключается в том, что автоматы над деревьями могут хранить только конечный объем информации в состояниях, и потому не точно связывают компоненты кортежей инвариантов.

Свойство 8. *Существуют инварианты, представимые элементарно, но не представимые регулярно.*

Доказательство. Покажем, что $ELEM \setminus REG \neq \emptyset$. В работе [57] показано, что $Diag \notin REG$.

Однако $Diag \in ELEM$, т.к. есть инвариант $I(x, y) \Leftrightarrow x = y$, $diseq(x, y) \Leftrightarrow \neg(x = y)$. $\square$

Ограничения на размер термов также позволяют представлять нерегулярные (и неэлементарные) языки.

Пример 6 (*LtGt*). Рассмотрим следующую систему дизъюнктов $LtGt$, описывающую простое свойство сравнений чисел Пеано.

$$\begin{aligned} lt(x, y) &\leftarrow x = Z \wedge y = S(y') \\ lt(x, y) &\leftarrow x = S(x') \wedge y = S(y') \wedge lt(x', y') \\ gt(x, y) &\leftarrow x = S(x') \wedge y = Z \\ gt(x, y) &\leftarrow x = S(x') \wedge y = S(y') \wedge gt(x', y') \\ \perp &\leftarrow lt(x, y) \wedge gt(x, y) \end{aligned}$$

Свойство 9. *Существуют инварианты, представимые элементарными формулами с ограничениями размера термов, но не представимые регулярно.*

Доказательство. Покажем, что $SIZEELEM \setminus REG \neq \emptyset$. Система $LtGt \in SIZEELEM$, потому что она имеет инвариант, представимый в классе $SIZEELEM$ следующим образом.

$$\begin{aligned} lt(x, y) &\Leftrightarrow size(x) < size(y) \\ gt(x, y) &\Leftrightarrow size(x) > size(y) \end{aligned}$$

Теперь необходимо показать, что порядки не представимы в классе REG. Предположим, что lt и gt имеют соответствующие инварианты: регулярные языки L и G . Поскольку регулярные языки закрыты относительно объединения, $L \cup G = \text{Diag} = \{(x, y) \mid x \neq y\}$ также должен быть регулярным, но это не так (см. свойство 8). $\square$

Подводя итоги Отметим, что классы REG и ELEM, а также классы REG и SIZEELEM не сопоставимы, то есть не включают друг друга. Тем самым представления, основанные на языках первого порядка, а также представление на основе автоматов над деревьями имеют фундаментальные различия. Хотя автоматы позволяют определять точные свойства термов на неограниченной глубине, они не могут представлять *реляционные свойства* термов, например, неравенство термов или порядки чисел Пеано. С другой стороны, хотя элементарные представления хороши для выражения отношений равенства и неравенства термов, однако они могут быть точными только на ограниченной глубине.



Рис. 3: Построение регулярной модели системы дизъюнктов Хорна над АТД.

6. Подход к выводу регулярных инвариантов

В этом разделе показано, как строить регулярные инварианты систем дизъюнктов Хорна над АТД, пользуясь инструментами поиска конечных моделей, такими как MACE [41], FINDER [51], PARADOX [10], CVC4 [20]. Основная схема предлагаемого подхода к выводу регулярных инвариантов представлена на рисунке 3. Кратко опишем эту схему.

Исходная система дизъюнктов Хорна над АТД может быть выражена формулой без равенства в отрицательном контексте, тестеров и селекторов. Затем каждый АТД-конструктор заменяется на новый неинтерпретируемый символ, и таким образом формула из теории АТД транслируется в формулу теории неинтерпретированных функций с равенством (EUF, equality and uninterpreted functions). Полученная формула передается в инструмент поиска конечных моделей. Наконец, при помощи соответствия между конечными моделями и автоматами над деревьями, мы получаем автомат, представляющий регулярный инвариант исходной системы.

6.1. Трансляция в теорию неинтерпретированных функций

Основная идея такой трансляции заключается в том, что на дизъюнкты над АТД можно посмотреть как на обычные формулы первого порядка в теории неинтерпретированных функций. По конечным моделям таких формул можно получать регулярные модели исходных систем дизъюнктов.

Так, по системе дизъюнктов можно построить другую систему путём замены всех конструкторов алгебраических типов неинтерпретированными функциональными символами. Однако интерпретация некоторых конструкторов может нарушать аксиомы АТД (неравенство термов, построенных при помощи разных конструкторов, инъективность, исчерываемость и т.д.). Полученная система с неинтерпретированными символами, подобно обычной формуле первого порядка, является либо выполнимой, либо нет. Если она выполнима, то каждому дизъюнкту соответствует некоторая структура $\mathcal{M}$, которую мы можем использовать для того, чтобы расширить эрбрановскую модель $\mathcal{H}$ интерпретациями неинтерпретированных предикатных символов, удовлетворяющих исходной системе над $\mathcal{H}$.

Например, для построения модели системы дизъюнктов в примере *Even* необходимо проверить выполнимость следующей формулы, являющейся формулой первого порядка над неинтерпретированными функциями.

$$\begin{aligned} & \forall x.(x = Z \rightarrow \text{even}(x)) \wedge \\ & \forall x, y.(x = S(S(y)) \wedge \text{even}(y) \rightarrow \text{even}(x)) \wedge \\ & \forall x, y.(\text{even}(x) \wedge \text{even}(y) \wedge y = S(x) \rightarrow \perp) \end{aligned} \quad (1)$$

Легко видеть, что приведённая формула первого порядка (1) выполняется следующей конечной моделью $\mathcal{M}$.

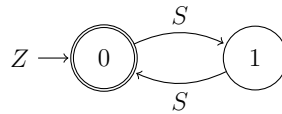
$$\begin{aligned} |\mathcal{M}|_{Nat} &= \{0, 1\} \\ \mathcal{M}(Z) &= 0 \\ \mathcal{M}(S)(x) &= 1 - x \\ \mathcal{M}(\text{even}) &= \{0\} \end{aligned}$$

6.2. Построение автоматов над деревьями по конечным моделям

Существует взаимно-однозначное соответствие между конечными моделями и автоматами над деревьями [38], на основе которого можно сконструировать процедуру для построения автоматов над деревьями (и, следовательно, регулярных моделей) по конечным моделям.

По конечной модели $\mathcal{M}$ для каждого предикатного символа $P \in \Sigma_P$ строится автомат $\mathcal{A}_P = (|\mathcal{M}|, \Sigma_F, \mathcal{M}(P), \tau)$. Для всех автоматов определяется общее множество переходов τ : для каждого $f \in \Sigma_F$ с арностью $\sigma_1 \times \dots \times \sigma_n \mapsto \sigma$ и для каждого $x_i \in |\mathcal{M}|_{\sigma_i}$ положим $\tau(f(x_1, \dots, x_n)) = \mathcal{M}(f)(x_1, \dots, x_n)$.

Таким образом, для примера *Even* определён следующий автомат $\mathcal{A}_{\text{Even}}$, изоморфный представленному в примере 2.



Теорема 1. Для каждого так построенного автомата $\mathcal{A}_P = (S, \Sigma_F, S_F, \tau)$ справедливо следующее утверждение:

$$L(\mathcal{A}_P) = \{\langle t_1, \dots, t_n \rangle \mid \langle \mathcal{M} \llbracket t_1 \rrbracket, \dots, \mathcal{M} \llbracket t_n \rrbracket \rangle \in \mathcal{M}(P)\}.$$

Доказательство. Очевидным образом следует из того, что $\mathcal{A}_P$ воспроизводит проверку выполнимости в $\mathcal{M}$. □

На практике это означает, что модель системы дизъюнктов Хорна над АДТ можно построить при помощи инструментов поиска конечных моделей, таких как MACE4 [41],

FINDER [51], PARADOX [10] или CVC4 (в специальном режиме [20]). И далее, если найдена *конечная* модель, то существует *регулярная эрбрановская модель* системы дизъюнктов.

6.3. Эрбрановские модели без равенства

Для доказательства корректности подхода необходимо показать, что эрбрановская модель, интерпретирующая предикаты автоматами над деревьями, является моделью исходной системы дизъюнктов Хорна. Это легко показать для систем дизъюнктов, не содержащих равенств в негативном контексте в теле (под нечётным числом отрицаний).

Пусть сигнатура Σ языка ограничений не содержит символа равенства. Других предикатных символов в нём нет, и потому можно предположить, что каждое ограничение в каждом дизъюнкте тождественно $\top$. Приведенный выше пример может быть переписан для соответствия этому требованию следующим образом.

$$\begin{aligned} \text{even}(Z) &\leftarrow \top \\ \text{even}(S(S(x))) &\leftarrow \text{even}(x) \\ \perp &\leftarrow \text{even}(x) \wedge \text{even}(S(x)) \end{aligned}$$

Лемма 3. Пусть система дизъюнктов Хорна $\mathcal{S}$ с неинтерпретированными предикатами $\mathcal{R} = \{P_1, \dots, P_k\}$ и без ограничений в телах дизъюнктов выполняется в некоторой модели $\mathcal{M}$, т.е. $\mathcal{M} \models C$ для всех $C \in \mathcal{S}$. Пусть также справедливо следующее:

$$X_i \stackrel{\text{def}}{=} \{\langle t_1, \dots, t_n \rangle \mid \langle \mathcal{M}[\![t_1]\!], \dots, \mathcal{M}[\![t_n]\!] \rangle \in \mathcal{M}(P_i)\}.$$

Тогда $\langle \mathcal{H}, X_1, \dots, X_k \rangle$ является эрбрановской моделью $\mathcal{S}$.

Доказательство. Так как по условию ограничения в телах дизъюнктов отсутствуют, то все дизъюнкты имеют вид $C \equiv R_1(\bar{t}_1) \wedge \dots \wedge R_m(\bar{t}_m) \rightarrow H$. Это означает, что выполнено следующее:

$$\langle \mathcal{H}, X_1, \dots, X_k \rangle \models C \iff \mathcal{M} \models C,$$

то есть каждый дизъюнкт из $\mathcal{S}$ выполняется в $\langle \mathcal{H}, X_1, \dots, X_k \rangle$. □

Например, для системы выше можно взять $X \stackrel{\text{def}}{=} \{t \mid \mathcal{M}[\![t]\!] = 0\} = \{S^{2n}(Z) \mid n \geq 0\}$, что даст модель системы.

6.4. Эрбрановские модели с равенством

Приведённые выше шаги позволяют строить регулярные инварианты систем дизъюнктов без равенств в отрицательном контексте. Рассмотрим, что произойдёт, если те же шаги применить к системе дизъюнктов с такими равенствами. Рассмотрим систему, состоящую из единственного дизъюнкта $Z \neq S(Z) \rightarrow \perp$. Очевидно, что эта

система невыполнима, так как $\mathcal{H} \models Z \neq S(Z)$. Однако, если рассматривать Z и S как неинтерпретированные функции, то этот дизъюнкт оказывается выполним, например, в модели $|\mathcal{M}|_{nat} = \{0\}$, $\mathcal{M}(Z) = \mathcal{M}(S)(*) = 0$. Таким образом, каждый дизъюнкт, имеющий неравенство в посылке, может быть выполнен, если сделать неравенство ложным, например, установив мощность конечной модели равной единице.

Эта проблема может быть решена следующим образом. Для каждого алгебраического типа (C, σ) можно ввести новый неинтерпретированный символ $diseq_\sigma$ и добавить его в множество реляционных символов $\mathcal{R}' \stackrel{\text{def}}{=} \mathcal{R} \cup \{diseq_\sigma \mid \sigma \in \Sigma_S\}$. Далее можно рассмотреть изменённую систему дизъюнктов $\mathcal{S}'$ над $\mathcal{R}'$, полученную так.

Без ограничения общности можно предположить, что ограничение каждого дизъюнкта $C \in \mathcal{S}$ содержит отрицания только над атомами. Тогда дизъюнкт C' можно получить из C путём замены каждого литерала вида $\neg(t =_\sigma u)$ (которое мы называем *ограничением неравенства*) атомарной формулой $diseq_\sigma(t, u)$. Для каждого предложения $C \in \mathcal{S}$ в $\mathcal{S}'$ добавляется C' . Наконец, для каждого алгебраического типа (C, σ) в $\mathcal{S}'$ добавляются следующие правила для $diseq_\sigma$:

$$\top \rightarrow diseq_\sigma(c(\bar{x}), c'(\bar{x}')) \text{ для всех различных конструкторов } c, c' \text{ сорта } \sigma.$$

и

$$diseq_{\sigma'}(x, y) \rightarrow diseq_\sigma(c(\dots, \underbrace{x}_{i\text{-ая позиция}}, \dots), c(\dots, \underbrace{y}_{i\text{-ая позиция}}, \dots))$$

для всех конструкторов c сорта σ , всех i и всех x, y сорта σ' .

Для каждого сорта σ в Σ_S обозначим $\mathcal{D}_\sigma \stackrel{\text{def}}{=} \{(x, y) \in |\mathcal{H}|_\sigma^2 \mid x \neq y\}$.

Хорошо известно, что универсально замкнутые дизъюнкты Хорна имеют наименьшую модель, которая является денотационной семантикой программы, моделируемой системой дизъюнктов [4]. Эта наименьшая модель является наименьшей неподвижной точкой оператора перехода. Из этого тривиально следует следующая лемма.

Лемма 4. *Правила для $diseq_\sigma$ имеют наименьшую модель над $\mathcal{H}$, которая интерпретирует символы $diseq_\sigma$ отношениями $\mathcal{D}_\sigma$.*

Простым следствием предыдущей леммы является следующий факт.

Лемма 5. *Для системы дизъюнктов Хорна $\mathcal{S}'$, полученной из некоторой другой при помощи описанной выше трансформации, если $\langle \mathcal{H}, X_1, \dots, X_k, Y_1, \dots, Y_n \rangle \models \mathcal{S}'$, то $\langle \mathcal{H}, X_1, \dots, X_k, \mathcal{D}_{\sigma_1}, \dots, \mathcal{D}_{\sigma_n} \rangle \models \mathcal{S}'$ (где отношения Y_i и $\mathcal{D}_{\sigma_i}$ интерпретируют предикатные символы $diseq_{\sigma_i}$).*

Пример 7. Система дизъюнктов $\mathcal{S} = \{Z \neq S(Z) \rightarrow \perp\}$ трансформируется в следующую

щую систему $\mathcal{S}'$.

$$\begin{aligned} \top &\rightarrow \text{diseq}_{\text{Nat}}(Z, S(x)) \\ \top &\rightarrow \text{diseq}_{\text{Nat}}(S(x), Z) \\ \text{diseq}_{\text{Nat}}(x, y) &\rightarrow \text{diseq}_{\text{Nat}}(S(x), S(y)) \\ \text{diseq}_{\text{Nat}}(Z, S(Z)) &\rightarrow \perp \end{aligned}$$

Система $\mathcal{S}$ из этого примера выполнима как формула первого порядка над неинтерпретированными функциями, но не выполняется в $\mathcal{H}$. Полученная же из неё с помощью описанной выше трансформации система $\mathcal{S}'$ не выполняется как формула первого порядка над неинтерпретированными функциями, поскольку тело запроса выводится из первого правила. Тем самым, приведённая трансформация помогает решить описанную проблему с равенством в отрицательном контексте. Таким образом, для построения регулярных моделей необходимо *искать конечные модели $\mathcal{S}'$ вместо $\mathcal{S}$* , а затем строить по ним автоматы над деревьями, как было описано в прошлом разделе.

Корректность всего подхода, приведённом в данном разделе, доказывается в следующей теореме.

Теорема 2. *Пусть $\mathcal{S}$ — система дизъюнктов Хорна, а $\mathcal{S}'$ — та же система дизъюнктов с заменёнными ограничениями неравенства. Если существует конечная модель системы $\mathcal{S}'$ как формулы в теории неинтерпретированных функций, то существует и регулярная эрбрановская модель исходной системы $\mathcal{S}$.*

Доказательство. Без ограничения общности можно предположить, что каждый дизъюнкт $C \in \mathcal{S}$ имеет следующий вид:

$$C \equiv u_1 \neq t_1 \wedge \dots \wedge u_k \neq t_k \wedge R_1(\bar{u}_1) \wedge \dots \wedge R_m(\bar{u}_m) \rightarrow H.$$

В $\mathcal{S}'$ этот дизъюнкт трансформируется в следующий дизъюнкт:

$$C' \equiv \text{diseq}(u_1, t_1) \wedge \dots \wedge \text{diseq}(u_k, t_k) \wedge R_1(\bar{u}_1) \wedge \dots \wedge R_m(\bar{u}_m) \rightarrow H.$$

Таким образом, каждое предложение в $\mathcal{S}'$ не содержит равенств (правила *diseq* также не содержат равенств), и по лемме 3 существует модель $\langle \mathcal{H}, X_1, \dots, X_k, U_1, \dots, U_n \rangle$ всех $C' \in \mathcal{S}'$. Тогда по лемме 5 имеем $\langle \mathcal{H}, X_1, \dots, X_k, \mathcal{D}_{\sigma_1}, \dots, \mathcal{D}_{\sigma_n} \rangle \models C'$. Однако очевидно следующее:

$$\langle \mathcal{H}, X_1, \dots, X_k, \mathcal{D}_{\sigma_1}, \dots, \mathcal{D}_{\sigma_n} \rangle \llbracket C' \rrbracket = \langle \mathcal{H}, X_1, \dots, X_k \rangle \llbracket C \rrbracket.$$

Но это означает, что $\langle \mathcal{H}, X_1, \dots, X_k \rangle \models C$ для каждого $C \in \mathcal{S}$. □

Замечание о существовании конечных моделей для ограничений неравенства. Можно показать, что если существует бесконечное число термов с сортом σ , то система дизъюнктов $\text{diseq}_{\sigma}(x, x) \rightarrow \perp$ выполняется только в бесконечных моделях. Другими словами, если заставить интерпретации *diseq* пропускать пары равных

термов, то система *не будет иметь конечных моделей*. Для сравнения, если заставить *diseq* быть ложным только на одной паре термов, то конечная модель может существовать. Например, для типа данных *Nat* такой запрос $diseq_{\sigma}(Z, Z) \rightarrow \perp$ имеет следующую конечную модель:

$$|\mathcal{M}|_{Nat} = \{0, 1\}, \mathcal{M}(Z) = 0, \mathcal{M}(S)(*) = 1, \mathcal{M}(diseq_{Nat}) = \{(0, 1), (1, 0), (1, 1)\}.$$

Интуитивно понятно, что с увеличением числа предполагаемых неравными термов уменьшается вероятность существования конечной модели. На практике это означает, что системы дизъюнктов, содержащие ограничения неравенств, с меньшей вероятностью будут выполнимы в конечных моделях.

6.5. Устранение тестеров и селекторов

На практике условия верификации могут содержать не только конструкторы алгебраических типов данных, но также тестеры (предикаты, проверяющие, что терм создан при помощи заданного конструктора) и селекторы (функции, получающие элементы, при помощи которых был сконструирован терм). Инструмент поиска конечных моделей интерпретирует свои входные данные как формулу в теории EUF, то есть работает в свободном домене, в котором могут нарушаться аксиомы АД для тестеров и селекторов. Подобно проблеме с равенством в отрицательном контексте, которая была решена путём введения новых дизъюнктов, мы можем решить эту проблему, предварительно заменив тестеры и селекторы новыми предикатами и добавив соответствующие дизъюнкты, фиксирующие их семантику. Например, это можно сделать для АД LISP-списков, определённых следующим образом:

$$List ::= nil \mid cons(car : \sigma, cdr : List).$$

Этот тип данных определяется конструкторами *nil*, *cons*, тестерами *nil?*, *cons?* и селекторами *car*, *cdr*. Для него следует ввести дизъюнкты, представленные ниже.

$$x = cons(r, y) \rightarrow car(x, r)$$

$$x = cons(y, r) \rightarrow cdr(x, r)$$

$$x = cons(y, z) \rightarrow cons?(x)$$

$$x = nil \rightarrow nil?(x)$$

7. Сравнения и соотнесения

В данном разделе проведено сравнение и соотнесение предложенного подхода к выводу инвариантов программ с алгебраическими типами данных с существующими подходами, реализованными в инструментах Z3 и ELDARICA. Подходы, реализованные в этих инструментах, основываются на элементарных представлениях инвариантов: на элементарном представлении и элементарном представлении с ограничениями размера термов соответственно. Также в данном разделе приведены альтернативные подходы к верификации программ с алгебраическими типами данных, не основанные на выводе инвариантов и потому не имеющие для них представления.

7.1. Инструмент Z3

Инструмент Z3 строит элементарные модели (класс ELEM из раздела 5.1). Этот инструмент построен на базе классической разрешающей процедуры для АТД, процедуры интерполяции и устранения кванторов [3]. Ядро инструмента, подход SPACER, основан на технике, называемой *достижимость, направляемая свойством* (property-directed reachability, PDR), которая равномерно распределяет время анализа между поиском контрпримеров и построением безопасного индуктивного инварианта, распространяя факты о достижимости небезопасных свойств и частичные леммы о безопасности.

7.2. Инструмент Eldarica

Инструмент ELDARICA строит модели с ограничениями размера термов, которые подсчитывают общее количество вхождений в них конструкторов (SIZEELEM из раздела 5.2). Он использует встроенный SMT-решатель PRINCESS [50], который предоставляет разрешающую процедуру и процедуру интерполяции для АТД с ограничениями размера термов, которые работают путём сведения этой теории к комбинации теорий неинтерпретируемых функций и линейной арифметики [26].

7.3. Другие инструменты с поддержкой АТД

Инструмент CVC4 в режиме индукции [48] (обозначается CVC4-IND²) использует ряд методов индуктивного рассуждения в рамках классического SMT-подхода. Индуктивная техника, применяемая в CVC4, глубоко интегрирована в SMT-решатель: она реализует сколемизацию с индуктивным усилением и подсчётом термов для поиска дочерних целей.

²Данный режим реализуется запуском инструмента CVC4 со следующими флагами: `quant-ind`, `quant-cf`, `conjecture-gen`, `conjecture-gen-per-round=3`, `full-saturate-quant`.

Расширение инструмента VERIMAP под названием VERIMAP-IDDT [52] обрабатывает условия проверки над теориями линейной арифметики и АТД и полностью устраняет АТД из исходной системы дизъюнктов, применяя методы сворачивания-разворачивания (fold/unfold). Из-за этой трансформации крайне трудоёмко по инварианту трансформированной системы построить инвариант исходной системы.

Также были предложены несколько работ для автоматизации индукции над АТД. В работе [59] предложен подход, основанный на синтаксически-направляемом синтезе [54], который строит индуктивные инварианты путём порождения вспомогательных лемм на основе неудачных подцелей доказательств и пользовательских шаблонов.

В работе [24] предложен фреймворк абстрактной интерпретации в регулярном домене для вывода инвариантов для функциональных программ высшего порядка над АТД. Авторы вводят систему типов, в которой каждый тип является регулярным языком, и используют подход CEGAR (Counterexample-Guided Abstraction Refinement) [12] для вывода регулярных инвариантов. Процедура в этой работе намного сложнее представленной в текущей работе, потому что она поддерживает функции высшего порядка, что не является целью данной работы, направленной на вывод инвариантов над чистой теорией АТД. Сужая контекст до функций первого порядка над АТД, мы получаем более простую процедуру вывода инвариантов, используя эффективные инструменты поиска конечных моделей.

8. Реализация и эксперименты

В данном разделе приведено описание реализации предложенного подхода по выводу регулярных инвариантов программ с алгебраическими типами данных. Также приведены результаты эмпирического сравнения реализованного инструмента с инструментами вывода инвариантов Z3 и ELDARICA, представляющих инварианты в классах ELEM (см. раздел 5.1) и SIZEELEM (см. раздел 5.2) соответственно.

8.1. Пилотная реализация

Предложенный подход (см. раздел 6) был реализован на языке F# в инструменте вывода регулярных инвариантов под названием RINGEN³. Инструмент RINGEN принимает на вход формулы в формате SMTLIB2 [1] и его TIP-расширении с конструкцией `define-fun-rec` [55]. Инструмент принимает на вход условия верификации программы и её свойство и проверяет, выполняется ли данное свойство, возвращая результат SAT и безопасный индуктивный инвариант, если оно выполняется, или завершается с результатом UNSAT, если нет. Таким образом, RINGEN может быть использован в качестве стороннего решателя для верификаторов функциональных программ, таких как MoCHi [32] и RCAML [58]. Инструмент RINGEN использует инструмент CVC4-f⁴ в качестве стороннего многосортного инструмента поиска конечных моделей для построения регулярных инвариантов (REG в наших обозначениях, см. раздел 4). Помимо регулярных инвариантов, подход к поиску конечных моделей в инструменте CVC4-f [20], основанный на инстанцировании кванторов, позволяет также находить контрпримеры к свойству, что даёт возможность доказывать небезопасность системы.

8.2. Выбор инструментов для экспериментов

В качестве инструментов для сравнения были выбраны Z3 и ELDARICA — единственные существующие на данный момент Хорн-решатели с поддержкой алгебраических типов данных. Также для справки были добавлены CVC4-IND и VERIMAP-IDDT — инструменты, предложенные авторами статей, из которых были взяты наборы данных для экспериментов. Последние два инструмента проверяют выполнимость систем, но при этом не строят инварианты *явно* в каком-либо представлении, и потому не находятся в фокусе данной работы, направленной на анализ различных представлений инвариантов.

³Regular Invariant Generator: <https://github.com/Columpio/RInGen>

⁴с опцией `--finite-model-find`

8.3. Набор данных

Эксперименты проводились на наборе данных *TIP* (Tons of Inductive Problems) [55] и двух наборах данных, полученных из набора данных из [52] следующим образом.

Набор данных из *TIP* состоит из индуктивных проблем над алгебраическими типами со списками, очередями, регулярными выражениями и целыми числами Пеано, изначально созданных из функциональных программ. В рамках данного исследования *TIP* был предобработан для обеспечения совместимости с инструментами, на которых проводились наши эксперименты. Предобработанная версия набора данных⁵ была также использована на соревнованиях Хорн-решателей СНС-COMP 2021⁶. Из *TIP* [55] были отобраны 454 проблем только с алгебраическими сортами (остальные проблемы содержали комбинации АД с другими теориями); все проблемы были преобразованы в системы дизъюнктов Хорна с ограничениями; неравенства были заменены атомами `diseq`, как описано в разделе 6.4, а все свободные сорта, объявленные через `(declare-sort ... 0)`, заменены на алгебраический тип чисел Пеано.

Набор данных [52] был модифицирован путём замены всех неалгебраических сортов на алгебраические — например, сорт *Int* из линейной арифметики был заменён на алгебраический тип целых чисел Пеано. Также в него были добавлены дизъюнкты-определения — например, арифметическое сложение было заменено сложением чисел Пеано, выраженным в дизъюнктах Хорна. Этот набор данных был разделён на два подмножества, обозначенных как *PosEq* и *Diseq*. *PosEq* — это набор систем дизъюнктов Хорна, в которых символ равенства появляется только в положительных контекстах в телах дизъюнктов. *Diseq* состоит из проблем с ограничениями неравенства в телах дизъюнктов, заменённых атомами `diseq` в соответствии с корректной трансформацией из раздела 6.4. В итоге, из этого набора данных было взято 60 систем дизъюнктов Хорна над двоичными деревьями, очередями, списками и числами Пеано.

Итоговый набор данных⁷ содержит 514 проблем. Он включает три набора данных *TIP*, *PosEq* и *Diseq*, и содержит задачи, полученные как из безопасных, так и из небезопасных программ с рекурсией.

Эксперименты проводились на компьютере с процессором Intel[®] Core[™] i7-4710HQ @ 2,50 ГГц и 2,50 ГГц, 16 ГБ оперативной памяти, под управлением ОС Arch Linux.

Для каждого инструмента было установлено ограничение на время исполнения в 300 секунд. Превышение ограничения на время исполнения подсчитывалось отдельно и количество таких случаев далее отдельно приводится.

⁵<https://github.com/chc-comp/ringen-adt-benchmarks>

⁶<https://chc-comp.github.io/>

⁷<https://gitlab.com/Columpio/adt-benchmarks/>

Представление инвариантов			REG	SizeELEM	ELEM	-	-
Набор данных	#	Результат	RINGEN	ELДАРICA	Z3	CVC4-IND	VERIMAP-IDDT
<i>PosEq</i>	35	SAT	27	1	4	0	0
<i>DisEq</i>	25	SAT	4	0	2	0	2
		UNSAT	1	1	1	1	1
<i>TIP</i>	454	SAT	30	46	26	0	16
		$\exists!$ SAT	13	25	7	0	0
		UNSAT	21	12	22	13	10
		$\exists!$ UNSAT	5	0	5	0	0
Total	514	SAT	61	47	32	0	18
		UNSAT	22	13	23	14	11

Таблица 1: Результаты экспериментов на трёх наборах данных с программами с алгебраическими типами данных. Число в каждой ячейке обозначает количество правильных результатов, полученных в пределах 300 секунд. Результат «SAT» обозначает, что система безопасна, результат «UNSAT» обозначает, что система небезопасна. В строках с обозначением $\exists!$ приведено число уникальных результатов, в которых инструмент из соответствующего столбца дал ответ, а другие нет. RINGEN использовался для построения *регулярных инвариантов*, Z3 использовался для построения *элементарных инвариантов*, ELДАРICA использовалась для построения *элементарных инвариантов с ограничениями размера термов*. CVC4-IND и VERIMAP-IDDT включены для справки.

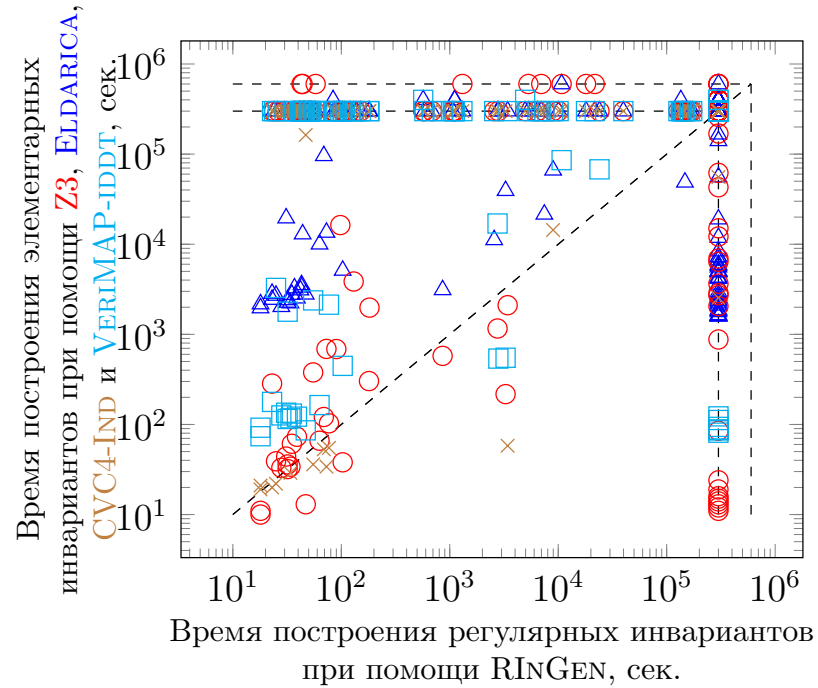


Рис. 4: Сравнение производительности инструментов. Каждая точка на графике представляет пару длительностей выполнения (сек × сек) инструмента RINGEN, строящего инварианты в классе REG (ось абсцисс), и другого инструмента, строящего инварианты в классе (SIZE)ELEM (ось ординат). Превышения ограничения времени размещаются на внутренних пунктирных линиях; ошибочные завершения размещаются на внешних пунктирных линиях.

8.4. Результаты экспериментов

Результаты экспериментов представлены в таблице 1.

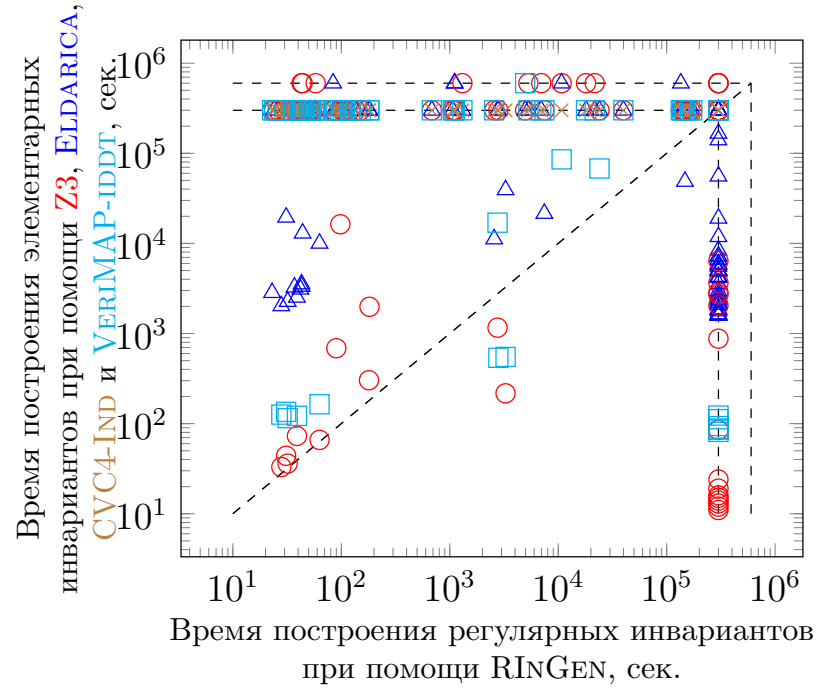


Рис. 5: Сравнение производительности инструментов *только на результатах «SAT»*. Тест включается в этот график, если хотя бы один из инструментов обнаружил инвариант.

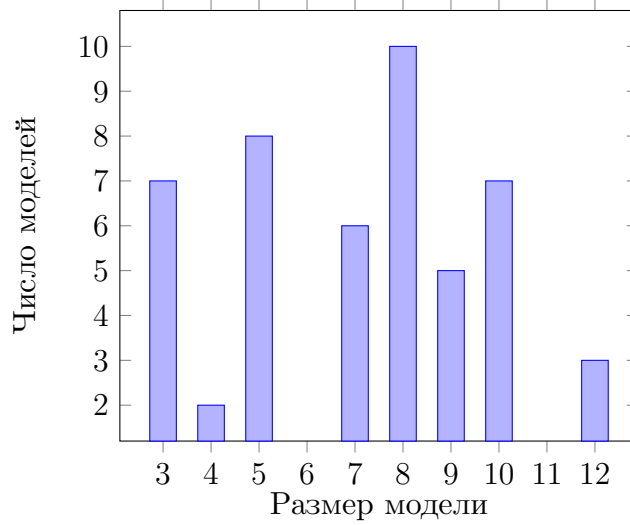


Рис. 6: Размеры конечных моделей, построенных инструментом CVC4-f — сторонним решателем, вызываемым инструментом RINGEN. Размер модели (на оси абсцисс) считается как *сумма* мощностей всех сортов.

На наборах данных *PosEq* и *Diseq* инструмент Z3 решил 7 задач, для остальных же задач вернул 8 UNKNOWN результатов и 45 раз превысил лимит времени. Инструмент ELDARICA решил 2 задачи (обе они были также решены Z3), а на остальных 58 задачах превысил лимит времени. RINGEN нашёл 31 регулярную модель, один контрпример и превысил лимит времени на оставшихся 28 задачах. Большинство из

решённых им задач находятся в наборе данных *PosEq*, который не содержит равенств в отрицательном контексте. Такие проблемы с большей вероятностью имеют регулярные решения, как и указано в разделе 6.4. Достойно упоминания, что каждая решённая инструментами Z3 или ELDARICA задача была также решена и инструментом RINGEN. Если инструменту RINGEN не удалось доказать выполнимость за отведённое время, ни один из других инструментов также не справился.

На наборе данных, полученном из *TIP*, наблюдается иное поведение инструментов. Во-первых, все 12 задач, для которых инструмент ELDARICA вернул ответ «UNSAT», RINGEN выдал решения. Иными словами RINGEN добился большего успеха в поиске контрпримеров. Инструменты RINGEN и Z3 нашли контрпримеры для 21 и 22 систем дизъюнктов соответственно. Большинство из этих систем совпадает, хотя некоторые из них уникальны для каждого инструмента.

Z3 превысил ограничение по времени 393 раза. Также он завершился 13 раз с результатом «UNKNOWN». RINGEN превысил ограничение по времени 403 раза без каких-либо других ошибок, а ELDARICA остановилась превысив ограничение по времени 368 раза, а также 28 раз завершилась с ошибкой⁸.

Наконец, инструмент ELDARICA нашёл 46 инварианта в противовес 30 инвариантам, найденным RINGEN. Из них 11 инвариантов общие, на них RINGEN был на два порядка быстрее. RINGEN решил 13 уникальных (не решённых ELDARICA) задач, все они содержат предикаты, аналогичные чётности чисел Пеано (например, «длина конкатенации списков четна, если сумма длин списков четна»), что вполне соответствует теории (см. свойство 7). ELDARICA решила 25 уникальных (не решённых RINGEN) задач, все они содержат порядковые предикаты ($<$, $\leq$, $>$, $\geq$) на числах Пеано. Стоит отметить, что именно этот тип предикатов рассматривается в свойстве 9 в качестве отделяющего класс SIZEELEM от остальных классов. Они хорошо поддержаны в этом классе, так как благодаря ограничениям размера они транслируются в соответствующие предикаты арифметики Пресбургера.

Графики на рисунках 4 и 5 показывают, что инструмент RINGEN не только выводит больше инвариантов, но и работает быстрее, чем другие инструменты, в среднем, на один порядок. На рисунке 4 некоторые небезопасные системы проверялись быстрее инструментами CVC4-IND, VERIMAP-IDDT и Z3. Это может быть связано с более эффективной процедурой инстанцирования кванторов в CVC4-IND и более сбалансированным компромиссом между инвариантным выводом и поиском контрпримеров в ядре Z3 (который также вызывается инструментом VERIMAP-IDDT).

Диаграмма на рисунке 6 показывает размеры моделей, полученных инструментом поиска конечных моделей CVC4-f, используемым инструментом RINGEN, во время экспериментов.

⁸C сообщением: «Cannot handle general quantifiers in predicates at the moment».

8.5. Выводы

Предлагаемый подход лучше справился с набором данных, основанном на [52]. Это связано с выразительностью автоматов над деревьями для представления инвариантов и эффективностью механизма поиска конечных моделей в CVC4-f — стороннем инструменте поиска конечных моделей, используемых инструментом RINGEN.

В наборе данных *TIP* инструмент ELDARICA нашёл больше инвариантов, чем RINGEN, но анализ систем, решённых только ELDARICA, показал, что все такие случаи строятся исключительно на порядковых предикатах для чисел Пеано, которые легко обрабатываются ELDARICA путём сведения в линейную арифметику. На примерах, решённых обоими инструментами, RINGEN был, в среднем, быстрее. Тем не менее, большинство примеров из набора данных *TIP*, исходно полученных из задач верификации, в настоящее время не могут быть решены при помощи современных инструментов, на которых было проведено сравнение.

Итак, можно заключить, что автоматы над деревьями вполне перспективны для автоматической верификации программ над алгебраическими типами данных. Они позволяют выразить сложные свойства рекурсивных предикатов и могут эффективно строиться современными инструментами. Предложенная пилотная реализация смогла найти на 25% больше инвариантов, чем другие инструменты, и при этом была, в среднем, на один порядок быстрее.

Заключение

В рамках данной работы были получены следующие результаты.

- Предложено регулярное представление инвариантов программ на основе автоматов над деревьями термов алгебраических типов данных.
- Проведён сравнительный анализ предложенного представления с (i) представлением, основанном на языках первого порядка и (ii) представлением, основанном на языках первого порядка с ограничениями на размер термов; предложенное представление позволяет выражать свойства термов на неограниченной глубине, чего не позволяют существующие представления.
- Предложен подход к выводу регулярных инвариантов программ с алгебраическими типами данных на основе поиска конечных моделей формул первого порядка с равенством и неинтерпретированными функциями.
- Предложенный подход реализован на языке F#, проведено эмпирическое сравнение полученного инструмента с инструментами Z3 и ELDARICA, использующими представление инвариантов на основе языков первого порядка; эксперименты проводились на 514 рекурсивных программах с алгебраическими типами данных, взятых из двух известных статей; предложенный инструмент вывел на 25% больше инвариантов и был, в среднем, на порядок быстрее других инструментов.

В рамках дальнейшей работы планируется провести эксперименты на наборах данных (benchmarks), содержащих более сложные алгебраические типы, например, синтаксические деревья программ. Также планируется исследовать возможность расширения регулярных моделей и реализовать инструмент вывода инвариантов для расширенного класса. Перспективным направлением может быть построение автоматов путём обучения [15], т.к. этот подход может быть адаптирован в технику, направляемую свойством безопасности [29], что позволит балансировать вывод инвариантов и построение контрпримеров.

Регулярные отношения могут быть расширены на другие классы языков (см., например, [8, 9, 17, 22, 30, 36]). Другое возможное направление исследований — автоматизировать вывод индуктивных инвариантов, представимых в языках первого порядка с предикатами вхождения в регулярный язык [11]. Это представление обладает разрешающей процедурой проверки выполнимости, а также замкнуто относительно логических операций, и включает в себя классы регулярных и элементарных инвариантов.

Список литературы

- [1] The SMT-LIB Standard: Version 2.6 : Rep. / Department of Computer Science, The University of Iowa ; Executor: Clark Barrett, Pascal Fontaine, Cesare Tinelli : 2017. — Available at www.SMT-LIB.org.
- [2] Barrett Clark, Shikanian Igor, Tinelli Cesare. An abstract decision procedure for a theory of inductive data types // Journal on Satisfiability, Boolean Modeling and Computation. — 2007. — Vol. 3. — P. 21–46.
- [3] Bjørner Nikolaj, Janota Mikolás. Playing with Quantified Satisfaction. // LPAR (short papers). — 2015. — Vol. 35. — P. 15–27.
- [4] Bjørner Nikolaj, McMillan Ken, Rybalchenko Andrey. On solving universally quantified horn clauses // International Static Analysis Symposium / Springer. — 2013. — P. 105–125.
- [5] Bjørner Nikolaj, McMillan Kenneth L, Rybalchenko Andrey. Program Verification as Satisfiability Modulo Theories. // SMT@ IJCAR. — 2012. — Vol. 20. — P. 3–11.
- [6] CVC4 / Clark Barrett, Christopher L. Conway, Morgan Deters et al. // Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11) / Ed. by Ganesh Gopalakrishnan, Shaz Qadeer. — Vol. 6806 of Lecture Notes in Computer Science. — Springer, 2011. — jul. — P. 171–177. — Snowbird, Utah.
- [7] Caferra Ricardo, Leitsch Alexander, Peltier Nicolas. Automated model building. — Springer Science & Business Media, 2013. — Vol. 31.
- [8] Synchronized-context free tree-tuple languages : Rep. / Citeseer ; Executor: Jacques Chabin, Jing Chen, Pierre Réty : 2006.
- [9] Chabin Jacques, Réty Pierre. Visibly pushdown languages and term rewriting // International Symposium on Frontiers of Combining Systems / Springer. — 2007. — P. 252–266.
- [10] Claessen Koen, Sörensson Niklas. New Techniques that Improve MACE-Style Finite Model Finding // Proceedings of the CADE-19 Workshop: Model Computation-Principles, Algorithms, Applications / Citeseer. — 2003. — P. 11–27.
- [11] Comon Hubert, Delor Catherine. Equational formulas with membership constraints // Information and Computation. — 1994. — Vol. 112, no. 2. — P. 167–216.
- [12] Counterexample-guided abstraction refinement / Edmund Clarke, Orna Grumberg, Somesh Jha et al. // International Conference on Computer Aided Verification / Springer. — 2000. — P. 154–169.

- [13] Datatypes with shared selectors / Andrew Reynolds, Arjun Viswanathan, Haniel Barbosa et al. // International Joint Conference on Automated Reasoning / Springer. — 2018. — P. 591–608.
- [14] De Moura Leonardo, 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.
- [15] Drewes Frank. MAT learners for recognizable tree languages and tree series // Acta Cybernetica. — 2009. — Vol. 19, no. 2. — P. 249–274.
- [16] Dynamically discovering likely program invariants to support program evolution / Michael D Ernst, Jake Cockrell, William G Griswold, David Notkin // IEEE transactions on software engineering. — 2001. — Vol. 27, no. 2. — P. 99–123.
- [17] Engelfriet Joost, Maletti Andreas. Multiple context-free tree grammars and multi-component tree adjoining grammars // International Symposium on Fundamentals of Computation Theory / Springer. — 2017. — P. 217–229.
- [18] Fermüller Christian G, Pichler Reinhard. Model representation via contexts and implicit generalizations // International Conference on Automated Deduction / Springer. — 2005. — P. 409–423.
- [19] Fermüller Christian G, Pichler Reinhard. Model representation over finite and infinite signatures // Journal of Logic and Computation. — 2007. — Vol. 17, no. 3. — P. 453–477.
- [20] Finite model finding in SMT / Andrew Reynolds, Cesare Tinelli, Amit Goel, Sava Krstić // International Conference on Computer Aided Verification / Springer. — 2013. — P. 640–655.
- [21] Floyd Robert W. Assigning Meanings to Programs // Proceedings of Symposium on Applied Mathematics. — 1967. — No. 32.
- [22] Gouranton Valérie, Réty Pierre, Seidl Helmut. Synchronized tree languages revisited and new applications // International Conference on Foundations of Software Science and Computation Structures / Springer. — 2001. — P. 214–229.
- [23] Gramlich Bernhard, Pichler Reinhard. Algorithmic aspects of Herbrand models represented by ground atoms with ground equations // International Conference on Automated Deduction / Springer. — 2002. — P. 241–259.
- [24] Haudebourg Timothée, Genet Thomas, Jensen Thomas. Regular language type inference with term rewriting // Proceedings of the ACM on Programming Languages. — 2020. — Vol. 4, no. ICFP. — P. 1–29.

- [25] Hoare Charles Antony Richard. An Axiomatic Basis for Computer Programming // Communications of the ACM. — 1969. — Vol. 12, no. 10. — P. 576–580.
- [26] Hojjat Hossein, Rümmer Philipp. Deciding and interpolating algebraic data types by reduction // 2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC) / IEEE. — 2017. — P. 145–152.
- [27] Hojjat Hossein, Rümmer Philipp. The ELDARICA horn solver // 2018 Formal Methods in Computer Aided Design (FMCAD) / IEEE. — 2018. — P. 158–164.
- [28] Horn Clause Solvers for Program Verification / Nikolaj Bjørner, Arie Gurfinkel, Ken McMillan, Andrey Rybalchenko // Fields of Logic and Computation II. — Springer, 2015. — P. 24–51.
- [29] IC3-flipping the E in ICE / Yakir Vizel, Arie Gurfinkel, Sharon Shoham, Sharad Malik // International Conference on Verification, Model Checking, and Abstract Interpretation / Springer. — 2017. — P. 521–538.
- [30] Jacquemard Florent, Klay Francis, Vacher Camille. Rigid tree automata // International Conference on Language and Automata Theory and Applications / Springer. — 2009. — P. 446–457.
- [31] Kapur Deepak, Majumdar Rupak, Zarba Calogero G. Interpolation for Data Structures // Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering. — 2006. — P. 105–116.
- [32] Kobayashi Naoki, Sato Ryosuke, Unno Hiroshi. Predicate abstraction and CEGAR for higher-order model checking // ACM SIGPLAN Notices / ACM. — Vol. 46. — 2011. — P. 222–233.
- [33] 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.
- [34] Kostyukov Yurii, Mordvinov Dmitry, Fedyukovich Grigory. [Beyond the Elementary Representations of Program Invariants over Algebraic Data Types](#) // Proceedings of the 42nd ACM SIGPLAN Conference on Programming Language Design and Implementation. — 2021.
- [35] Kostyukov Yurii, Mordvinov Dmitry, Fedyukovich Grigory. Beyond the Elementary Representations of Program Invariants over Algebraic Data Types. — 2021. — 2104.04463.

- [36] Limet Sébastien, Réty Pierre, Seidl Helmut. Weakly regular relations and applications // International Conference on Rewriting Techniques and Applications / Springer. — 2001. — P. 185–200.
- [37] Lisitsa Alexei. Finite models vs tree automata in safety verification // 23rd International Conference on Rewriting Techniques and Applications (RTA'12) / Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. — 2012.
- [38] Matzinger Robert. Comparing computational representations of Herbrand models // Kurt Gödel Colloquium on Computational Logic and Proof Theory / Springer. — 1997. — P. 203–218.
- [39] Matzinger Robert. On computational representations of Herbrand models // Uwe Egly and Hans Tompits, editors. — 1998. — Vol. 13. — P. 86–95.
- [40] Matzinger Robert. Computational representations of models in first-order logic : Ph.D. thesis / Robert Matzinger ; Technische Universität Wien, Austria. — 2000.
- [41] McCune William. Mace4 Reference Manual and Guide // arXiv preprint cs/0310055. — 2003.
- [42] McMillan Kenneth L. Interpolation and SAT-based model checking // International Conference on Computer Aided Verification / Springer. — 2003. — P. 1–13.
- [43] Oppen Derek C. Reasoning about recursively defined data structures // Journal of the ACM (JACM). — 1980. — Vol. 27, no. 3. — P. 403–411.
- [44] Peltier Nicolas. Constructing infinite models represented by tree automata // Annals of Mathematics and Artificial Intelligence. — 2009. — Vol. 56, no. 1. — P. 65–85.
- [45] Pham Tuan-Hung, Gacek Andrew, Whalen Michael W. Reasoning About Algebraic Data Types with Abstractions // J. Autom. Reasoning. — 2016. — Vol. 57, no. 4. — P. 281–318.
- [46] Refinement types for Haskell / Niki Vazou, Eric L. Seidel, Ranjit Jhala et al. // ICFP. — ACM, 2014. — P. 269–282.
- [47] Reynolds Andrew, Blanchette Jasmin Christian. A decision procedure for (co) datatypes in SMT solvers // Journal of Automated Reasoning. — 2017. — Vol. 58, no. 3. — P. 341–362.
- [48] Reynolds Andrew, Kuncak Viktor. Induction for SMT solvers // International Workshop on Verification, Model Checking, and Abstract Interpretation / Springer. — 2015. — P. 80–98.

- [49] Reynolds John C. Separation logic: A logic for shared mutable data structures // Logic in Computer Science, 2002. Proceedings. 17th Annual IEEE Symposium on / IEEE. — 2002. — P. 55–74.
- [50] Rümmer Philipp. A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic // International Conference on Logic for Programming Artificial Intelligence and Reasoning / Springer. — 2008. — P. 274–289.
- [51] Slaney John. FINDER: Finite Domain Enumerator System Description // International Conference on Automated Deduction / Springer. — 1994. — P. 798–801.
- [52] Solving Horn Clauses on Inductive Data Types Without Induction / Emanuele De Angelis, Fabio Fioravanti, Alberto Pettorossi, Maurizio Proietti // Theory and Practice of Logic Programming. — 2018. — Vol. 18, no. 3-4. — P. 452–469.
- [53] Suter Philippe, Dotta Mirco, Kuncak Viktor. Decision Procedures for Algebraic Data Types with Abstractions // Acm Sigplan Notices. — 2010. — Vol. 45, no. 1. — P. 199–210.
- [54] Syntax-guided synthesis / Rajeev Alur, Rastislav Bodík, Garvit Juniwal et al. // FMCAD. — IEEE, 2013. — P. 1–17.
- [55] TIP: tons of inductive problems / Koen Claessen, Moa Johansson, Dan Rosén, Nicholas Smallbone // Conferences on Intelligent Computer Mathematics / Springer. — 2015. — P. 333–337.
- [56] Teucke Andreas, Voigt Marco, Weidenbach Christoph. On the expressivity and applicability of model representation formalisms // International Symposium on Frontiers of Combining Systems / Springer. — 2019. — P. 22–39.
- [57] Comon H., Dauchet M., Gilleron R. et al. Tree Automata Techniques and Applications. — Available on: <https://jacquema.gitlabpages.inria.fr/files/tata.pdf>. — 2008. — release November, 18th 2008.
- [58] Unno Hiroshi, Torii Sho, Sakamoto Hiroki. Automating induction for solving horn clauses // International Conference on Computer Aided Verification / Springer. — 2017. — P. 571–591.
- [59] Yang Weikun, Fedyukovich Grigory, Gupta Aarti. Lemma Synthesis for Automating Induction over Algebraic Data Types // International Conference on Principles and Practice of Constraint Programming / Springer. — 2019. — P. 600–617.
- [60] Zhang Ting, Sipma Henny B, Manna Zohar. Decision procedures for recursive data structures with integer constraints // International Joint Conference on Automated Reasoning / Springer. — 2004. — P. 152–167.