УНИВЕРСИТЕТ ИТМО
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ноябрь–декабрь 2020 Том 20 № 6 ISSN 2226-1494 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
November–December 2020 Vol. 20 No 6 ISSN 2226-1494 http://ntv.ifmo.ru/en/
НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК УНИВЕРСИТЕТ ИТМО ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

УДК 004.832.25 doi: 10.17586/2226-1494-2020-20-6-841-847

ПРИМЕНЕНИЕ ИНКРЕМЕНТАЛЬНЫХ SAT-РЕШАТЕЛЕЙ ДЛЯ РЕШЕНИЯ NP-ТРУДНЫХ ЗАДАЧ НА ПРИМЕРЕ ЗАДАЧИ СИНТЕЗА МИНИМАЛЬНЫХ БУЛЕВЫХ ФОРМУЛ

К.И. Чухарев

Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация
Адрес для переписки: kchukharev@itmo.ru

Информация о статье
Поступила в редакцию 02.10.20, принята к печати 20.10.20
Язык статьи — русский

Ссылка для цитирования: Чухарев К.И. Применение инкрементальных SAT-решателей для решения NP-трудных задач на примере задачи синтеза минимальных булевых формул // Научно-технический вестник информационных технологий, механики и оптики. 2020. Т. 20. № 6. С. 841–847. doi: 10.17586/2226-1494-2020-20-6-841-847

Аннотация

Предмет исследования. Рассмотрен метод решения NP-трудной задачи синтеза минимальной булевой формулы по заданной таблице истинности. Предложено решение указанной задачи, основанное на ее сведении к задаче выполнимости булевой формулы (SAT). Рассмотрены вопросы эффективной и удобной программной реализации сведений NP-трудных задач к SAT. **Метод.** Для решения задачи синтеза минимальной булевой формулы используется подход программирования в ограничениях: для заданной таблицы истинности строится SAT-формула, выполнимая тогда и только тогда, когда существует искомая булева формула заданного размера, удовлетворяющая заданной таблице. Отличительной особенностью разработанного метода является возможность использования инкрементальных программных средств для решения SAT (SAT-решателей). **Основные результаты.** Предложен метод синтеза минимальной по числу операторов и терминалов булевой формулы по заданной таблице истинности. Метод основан на сведении к SAT и позволяет использовать инкрементальные SAT-решатели. Разработан фреймворк kotlin-satlib, позволяющий эффективно и удобно использовать языки Kotlin и Java для программной реализации сведения различных NP-трудных задач к SAT, пользуясь нативным взаимодействием с SAT-решателями посредством технологии JNI. Предложенный метод синтеза минимальной булевой формулы реализован на языке программирования Kotlin с использованием разработанного фреймворка kotlin-satlib. **Практическая значимость.** Экспериментальное исследование на примере синтеза минимальной булевой формулы показало, что использование инкрементальных SAT-решателей для решения NP-трудных задач является целесообразным, поскольку позволяет уменьшить суммарное время решения задач по сравнению с использованием неинкрементального подхода.

Ключевые слова
задача выполнимости SAT, инкрементальные SAT-решатели, NP-трудные задачи, программирование в ограничениях, синтез булевых формул, преобразования Цейтина, нарушение симметрии, Kotlin, фреймворк, Java Native Interface

Благодарности
Исследование выполнено при финансовой поддержке JetBrains Research.

doi: 10.17586/2226-1494-2020-20-6-841-847

APPLICATION OF INCREMENTAL SATISFIABILITY PROBLEM SOLVERS FOR NON-DETERMINISTIC POLYNOMIAL-TIME HARD PROBLEMS AS ILLUSTRATED BY MINIMAL BOOLEAN FORMULA SYNTHESIS PROBLEM

K.I. Chukharev

ITMO University, Saint Petersburg, 197101, Russian Federation
Corresponding author: kchukharev@itmo.ru

Article info
Received 02.10.20, accepted 20.10.20
Article in Russian

For citation: Chukharev K.I. Application of incremental satisfiability problem solvers for non-deterministic polynomial-time hard problems as illustrated by minimal Boolean formula synthesis problem. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2020, vol. 20, no. 6, pp. 841–847 (in Russian). doi: 10.17586/2226-1494-2020-20-6-841-847

Abstract

Subject of Research. The paper considers a method for solution of the nondeterministic polynomial hard problem (NP-hard problem) of a minimal Boolean formula synthesis from a given truth table. The solution of this problem is proposed based on its reduction to the Boolean satisfiability problem (SAT). The issues of efficient and convenient software implementation are discussed for reducing nondeterministic polynomial hard problems to the satisfiability problem. **Method.** For a minimal Boolean formula synthesis, the constraint programming approach was used: a SAT-formula was created for a given truth table, satisfiable if and only if, there exists a Boolean formula of a given size that satisfies the given truth table. The developed method accent is the application of incremental satisfiability problem solvers. **Main Results.** A method is proposed for synthesis of a Boolean formula, minimal with respect to the number of operators and terminals, for a given truth table. The method is based on reducing to satisfiability problem and provides the usage of incremental satisfiability problem solvers. The kotlin-satlib framework is developed with the possibility to use the Kotlin and Java languages effectively and conveniently for the software implementation of reducing various nondeterministic polynomial-time hard problems to satisfiability problem. Native interaction with satisfiability problem solvers by Java Native Interface (JNI) technology is used. The proposed method for the minimal Boolean formula synthesis is implemented in the Kotlin programming language using the developed kotlin-satlib framework. **Practical Relevance.** An experimental study on the example of minimal Boolean formula synthesis has shown that the usage of incremental satisfiability problem solvers for nondeterministic polynomial hard problems is reasonable, since it reduces the total solving time in comparison with the non-incremental approach application.

Keywords

satisfiability problem (SAT), incremental satisfiability problem (SAT) solvers, non-deterministic polynomial-time (NP) hard problems, constraint programming, Boolean formula synthesis, Tseytin transformations, symmetry breaking, Kotlin, framework, Java Native Interface

Acknowledgements

The reported study was funded by JetBrains Research.

Введение

Задача синтеза булевой формулы заключается в построении логической формулы, зависящей от N переменных $x_1 \ldots x_N$ (возможно, не от всех, т. е. некоторые переменные могут не использоваться в полученной формуле), по заданной таблице истинности, с использованием заданных логических операций. Заданная таблица истинности может быть как полной (размера $2^{(2^N)}$), так и частичной — для некоторых наборов переменных (в дальнейшем также называемых «входами») значение логической функции может быть не определено. Соответствующая логическая функция имеет один логический выход, значения для которого на различных входах и записаны в таблице истинности. Стоит отметить, что список допустимых логических операций может варьироваться, также как и возможность применения операций к подвыражениям, в зависимости от желаемого результата (например, формулы в так называемой нормальной форме отрицания (negation normal form, NNF) могут содержать логическое отрицание, применяемое только к переменным, но не к комплексным выражениям). В настоящей работе рассматривается задача синтеза булевой формулы, содержащей следующие логические операции, без дополнительных ограничений на их применимость к подвыражениям: $\wedge$ (логическое И), $\vee$ (логическое ИЛИ) и $\neg$ (логическое отрицание).

Также рассматривается задача синтеза минимальной булевой формулы, т. е. формулы минимального размера, удовлетворяющей заданной таблице истинности. Несмотря на простоту формулировки, задача синтеза минимальной булевой формулы по полной или частичной таблице истинности является NP-трудной [1]. На практике данная задача обычно решается с помощью эвристических подходов, не гарантирующих минимального ответа. Наиболее часто используемым подходом является использование метода Espresso [2], реализация которого доступна в виде одноименного программного средства. Несмотря на то, что этот эвристический подход был разработан относительно давно, его успех до сих пор не был существенно преодолен — многие современные подходы в той или иной степени являются модификациями Espresso, например, BOOMII [3]. Метод Espresso позволяет синтезировать «минимальные» булевы формулы по заданным полным или частичным таблицам истинности, включая возможность синтезировать функции с множественными выходами. В процессе минимизации могут использоваться различные оптимизационные критерии, например, суммарное число логических вентилей или число использованных литералов. Отличительной особенностью данного метода является его высокая эффективность. Однако стоит отметить, что получаемое с помощью Espresso решение не является «точным», т. е. не является наименьшим — возможно существование меньшего решения, даже при использовании большого числа итераций. В тех случаях, когда требуется «точное» решение, т. е. наименьшая из возможных булевых формул, необходимо использование других подходов, например, программирование в ограничениях, а именно, сведение к задаче выполнимости.

Задача выполнимости (*Boolean Satisfiability Problem* — SAT) формулируется следующим образом: для заданной булевой формулы в конъюнктивной нормальной форме (КНФ) требуется определить, существует ли такая подстановка значений логических переменных, что формула становится истинной [4]. Формула находится в КНФ, если она является конъюнкцией дизъюнктов. Дизъюнкт (*clause*) — дизъюнкция литералов. Литерал — некоторая переменная или ее отрицание. Пример КНФ, состоящей из трех дизъюнктов: $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee x_3)$. Задача SAT является исторически первой задачей, для которой была доказана NP-полнота [5].

Традиционным способом решения других NP-полных и NP-трудных задач является их сведение к задаче SAT, так как такое сведение осуществляется за полиномиальное время, а задача SAT имеет достаточно простую формулировку и эффективные программные средства для ее решения — SAT-решатели, реализующие алгоритм CDCL (Conflict-Driven Clause Learning) [6].

Целью настоящей работы является исследование применимости инкрементальных SAT-решателей для решения NP-трудных задач. Рассматриваемая в работе задача синтеза минимальной булевой формулы заключается в нахождении формулы минимального размера, т. е. формулы с минимальным числом вершин в дереве разбора. Для решения этой задачи применяется программирование в ограничениях, а именно, сведение к задаче SAT. Для поиска минимального значения числа вершин применяются два подхода: итеративный (с перезапуском SAT-решателя на каждом шаге) и инкрементальный (с использованием предположений и без перезапуска SAT-решателя на каждом шаге). Для того чтобы иметь возможность провести экспериментальное сравнение этих подходов, в работе также предлагается фреймворк kotlin-satlib для моделирования сведений к SAT.

Предлагаемый подход к синтезу минимальной булевой формулы путем сведения к задаче SAT

Для логической формулы может быть построено дерево разбора — укорененное (rooted) дерево, во внутренних узлах которого находятся логические операции, а вершины-листья отмечены переменными $x_1 \dots x_N$. Связи между вершинами соответствуют применению соответствующих операций к вершинам-потомкам. Каждое поддерево такого дерева разбора может рассматриваться как некоторое подвыражение исходной формулы. Размер дерева разбора — число вершин, из которых оно состоит (включая вершины-листья). Размер логической формулы — размер соответствующего дерева разбора. В дальнейшем размер дерева разбора или булевой формулы будет обозначаться как P.

Сведение задачи к SAT обычно выглядит как декларативное описание с помощью логических переменных и ограничений структуры желаемого решения и его взаимодействия с исходными данными. В случае рассматриваемой задачи синтеза булевой формулы от N переменных, необходимо закодировать структуру дерева разбора синтезируемой формулы заданного размера P, а также логические значения каждой вершины дерева разбора (каждая вершина соответствует некоторому подвыражению; корень дерева соответствует всей формуле) на различных входах. После этого необходимо добавить ограничение на соответствие значений синтезируемой функции значениям в заданной таблице истинности. Полученную в результате такого сведения SAT-формулу (в КНФ) необходимо решить с помощью SAT-решателя для получения либо искомой булевой формулы заданного размера P, либо доказательства ее несуществования для заданного P.

Для нахождения минимальной булевой формулы необходимо каким-либо образом определить минимальное значение P, при котором решение существует. Для этого в данной работе используется перебор параметра P снизу вверх, начиная с единицы — таким образом, первое найденное решение будет минимальным из возможных. При этом используются два подхода:

1) итеративный подход, при котором SAT-решатель перезапускается на каждом шаге для каждого нового значения P;
2) инкрементальный подход, при котором на очередной итерации перебора параметра P сведение расширяется только теми ограничениями, которые зависят от нового значения P, а вызовы SAT-решателя производятся с использованием предположений (*assumptions*), что позволяет **не перезапускать** SAT-решатель даже после получения UNSAT — сообщения об отсутствии решения при заданных ограничениях.

Рассмотрим подробнее составляющие сведения к SAT. Параметр P отвечает за размер синтезируемой формулы — число вершин дерева разбора. Вершины дерева разбора нумеруются последовательно, начиная с корневой, имеющей индекс 1. В общем случае порядок индексации вершин не имеет значения, однако на практике использование нумерации вершин дерева в порядке BFS-обхода (Breadth-First Search — поиск в ширину) позволяет существенно сократить размер сведения, а также избавиться от большого числа изоморфных решений, что положительно влияет на эффективность метода — это так называемое «нарушение симметрий» [7], широко используемое при решении задач с помощью методов программирования в ограничениях. Для обеспечения BFS-нумерации необходимо, во-первых, чтобы для любой пары смежных вершин дерева (родитель-потомок) номер родительской вершины был меньше номера потомка; а во-вторых, чтобы вершины-потомки нумеровались по порядку, без пропусков индексов. В рассматриваемой задаче вершины могут иметь не более двух потомков — с номерами c и $(c + 1)$, где $c > p$.

Каждая вершина дерева может быть либо одной из допустимых логических операций ($\wedge$, $\vee$, $\neg$), либо терминалом, что кодируется с помощью переменной $\tau_p \in \{\wedge, \vee, \neg, \bot\}$, где $p \in [1 \dots P]$, а $\bot$ соответствует вершине-терминалу. Переменная $\chi_p \in [0 \dots N]$ кодирует номер переменной (от 1 до N), которой соответствует вершина p. Только вершины-терминалы имеют ассоциированные переменные: $(\tau_p = \bot) \leftrightarrow (\chi_p = 0)$.

Переменная $\pi_p \in [0 \dots (p - 1)]$, где $p \in [1 \dots P]$, кодирует номер родительской вершины для вершины p. Как было упомянуто выше, номер родителя при использовании BFS-нумерации должен быть меньше номера самой вершины p, поэтому доменом этой переменной является диапазон от 0 до $(p - 1)$. При этом $\pi_p = 0$ означает, что у вершины p в дереве нет родителя — это выполняется только для корневой вершины.

Переменная $\sigma_p \in \{0\} \cup [(p + 1) \dots P]$, где $p \in [1 \dots P]$, кодирует номер левого потомка вершины p. Взаимосвязь между переменными π и σ кодируется следующим образом: $(\sigma_p = c) \rightarrow (\pi_c = p)$. В том случае, если тип вершины p — терминал, то такая вершина не имеет потомков: $(\tau_p = \bot) \rightarrow (\sigma_p = 0)$. В том случае, если вер-

шина p имеет тип бинарной операции ($\wedge$ или $\vee$), то вершина с номером $(\sigma_p + 1)$ неявно считается правым потомком вершины p: $(\tau_p \in \{\wedge, \vee\}) \wedge (\sigma_p = c) \rightarrow (\pi_{c+1} = p)$.

Переменная $\vartheta_{p,u} \in \mathbb{B}$ ($p \in [1…P]$, $u \in U$) кодирует логическое значение вершины p на входе $u \in U$, где U — множество входов в заданной таблице истинности. Значение корневой вершины соответствует значению всей синтезируемой функции и должно совпадать со значением, указанным в заданной таблице истинности. Значения вершин рассчитываются исходя из их типа, что декларативно можно описать следующими ограничениями:

$$(\tau_p = \bot) \wedge (\chi_p = x) \rightarrow \wedge_{u \in U}(\vartheta_{p,u} \leftrightarrow u_x);$$
$$(\tau_p = \wedge) \wedge (\sigma_p = c) \rightarrow \wedge_{u \in U}(\vartheta_{p,u} \leftrightarrow \vartheta_{c,u} \wedge \vartheta_{c+1,u});$$
$$(\tau_p = \vee) \wedge (\sigma_p = c) \rightarrow \wedge_{u \in U}(\vartheta_{p,u} \leftrightarrow \vartheta_{c,u} \vee \vartheta_{c+1,u});$$
$$(\tau_p = \neg) \wedge (\sigma_p = c) \rightarrow \wedge_{u \in U}(\vartheta_{p,u} \leftrightarrow \vartheta_{c,u}).$$

Фреймворк kotlin-satlib

Для записи и передачи вышеописанного сведения в SAT-решатель в ходе выполнения данной работы разработан специализированный фреймворк kotlin-satlib[1]. Разработанный фреймворк выполнен в виде библиотеки на языке Kotlin и состоит из нескольких модулей: модуль взаимодействия с SAT-решателями через их нативные интерфейсы с помощью технологии JNI (Java Native Interface); модуль для упрощенной записи распространенных видов ограничений с использованием преобразований Цейтина [8]; модуль для манипуляции переменными с ограниченными доменами (например, целочисленными); модуль для манипуляции многомерными массивами SAT-переменных. В последующих разделах приведено описание этих модулей.

Модуль взаимодействия с SAT-решателями на основе технологии JNI

Практически все современные SAT-решатели предоставляют нативный программный интерфейс для взаимодействия с ними, однако, так как подавляющее большинство решателей написаны на языках C/C++ (ввиду строгих требований к эффективности реализации), их прямое использование в языках более высокого уровня весьма затруднено. В качестве целевой платформы используется JVM (Java Virtual Machine), а именно, языки программирования Java и Kotlin. Подавляющее большинство взаимодействий с нативным программным обеспечением из JVM так или иначе выполняется с помощью технологии JNI[2] (Java Native Interface). Данная технология позволяет описывать нативные методы (в Java — с помощью ключевого слова «native», в Kotlin — с помощью ключевого слова «external») в Java/Kotlin классах, реализация которых выполняется на нативном языке (обычно на C/C++), где можно получить доступ к другому нативному коду, например, к SAT-решателям. Полученные нативные реализации компилируются в динамические библиотеки (на GNU/Linux — *shared object library*, на Windows — *dynamic-link library*) и становятся доступными для виртуальной машины (JVM) во времени исполнения программы.

Стоит отметить существование библиотеки jnisat на языке Java — программной обертки для решателей PicoSAT [9] и MiniSAT [10], использующей описанную технологию JNI. Именно эта библиотека легла в основу фреймворка kotlin-satlib: собственная реализация выполнена на языке Kotlin с поддержкой вызова кода из Java, был добавлен общий унифицированный интерфейс для SAT-решателей, а также добавлена поддержка таких современных SAT-решателей, как CaDiCaL[3], Glucose [11] и cryptominisat [12].

Модуль записи ограничений с использованием преобразований Цейтина

Практически все ограничения, определенные в разделе про кодирование задачи синтеза булевой формулы, не были представлены в КНФ, что затрудняет их прямую передачу в SAT-решатель — предварительно необходимо конвертировать все ограничения в набор дизъюнктов КНФ. Однако стоит учитывать, что некоторые выражения, например, вида $(x_1 \wedge y_1) \vee \ldots \vee (x_N \vee y_N)$, при конвертации их в эквивалентные КНФ имеют экспоненциальный размер (по числу дизъюнктов) относительно исходного. Для решения этой проблемы можно воспользоваться неэквивалентными преобразованиями, которые сохраняют (не)выполнимость формулы, так как при решении задачи SAT интерес представляет только получаемая модель или же доказательство отсутствия решения. Одними из наиболее известных таких преобразований являются преобразования Цейтина [8], основная идея которых — введение дополнительных переменных, кодирующих и заменяющих собой логические вентили в формуле, постепенно сводя ее к КНФ, размер которой растет полиномиально. Например, для выражения $A \wedge B$ преобразование Цейтина вводит новую переменную $C \leftrightarrow A \wedge B$, кодируемую в КНФ следующим образом:

$$(\neg A \vee \neg B \vee C) \wedge (A \vee \neg C) \wedge (B \vee \neg C).$$

Добавление новых переменных, безусловно, увеличивает теоретическую сложность задачи, однако на практике добавление дополнительных структурных ограничений может даже помочь ускорить процесс решения. К тому же современные SAT-решатели способны манипулировать миллионами переменных, поэтому применение преобразований Цейтина в случаях, когда наивная конвертация ограничений в КНФ приводила бы к экспоненциальному росту размера задачи, является целесообразным.

Разработанный фреймворк kotlin-satlib содержит модуль Ops, производящий такие преобразования Цейтина автоматически. Модуль Ops также содержит множество типичных видов ограничений, например, функция implyIffOr(x1: Lit, x2: Lit, xs: Iterable<Lit>)

[1] https://github.com/Lipen/kotlin-satlib

[2] https://docs.oracle.com/javase/8/docs/technotes/guides/jni

[3] https://github.com/arminbiere/cadical

позволяет задать ограничение вида $x_1 \rightarrow (x_2 \leftrightarrow \vee_{x_i \in xs} x_i)$. Дополнительные перегрузки этих функций с альтернативными контейнерами используемых литералов, например, implyIffOr(x1: Lit, x2: Lit, vararg xs: Lit), образуют небольшой предметно-ориентированный язык (Domain Specific Language, DSL), позволяющий пользователю сконцентрироваться на моделировании задачи и использовать практически произвольные виды ограничений, а не на механических действиях, связанных с кодированием дополнительных переменных и конвертацией ограничений в КНФ, необходимую для SAT-решателя.

Модуль манипуляции переменными с ограниченным доменом

Большинство переменных, определенных в разделе про кодирование задачи синтеза булевой формулы, не были логическими, а имели либо целочисленное значение из некоторого известного диапазона, либо некоторое значение из заданного множества допустимых, например, $\tau_p \in \{\wedge, \vee, \neg, \bot\}$. Стоит сразу отметить, что SAT-решатели поддерживают только логические переменные, однако на практике моделирование исходной задачи с использованием переменных с ограниченными доменами, а не только логических, зачастую оказывается гораздо более выразительным и позволяет смотреть на задачу с интуитивной стороны. Основной подход к непосредственному кодированию переменных с ограниченным доменом — так называемое *onehot*-кодирование, при котором каждому возможному значению переменной из домена сопоставляется логическая переменная. Например, истинная логическая переменная $\tau'_{p,\wedge}$ соответствует ситуации, когда $\tau_p = \wedge$. При этом необходимо также добавить ограничение на то, что ровно один из *onehot*-литералов может иметь истинное значение — это может быть сделано в виде комбинации ограничений AtLeastOne (не менее одного) и AtMostOne (не более одного). Ограничение AtLeastOne является тривиальным и представляет собой один дизъюнкт, содержащий все объявленные литералы. А вот ограничение AtMostOne может быть закодировано множеством способов — от простых и интуитивных [13] до комплексных и эффективных [14], чему посвящены многочисленные исследования.

Стоит отметить, что *onehot*-кодирование — далеко не единственный способ представления переменных в виде набора литералов, хотя и самый интуитивный и удобный в большинстве ситуаций. Среди альтернативных способов кодирования можно выделить следующие [15]: порядковое (*order encoding*) [16], бинарное (*binary encoding*), комбинированное (*onehot* + *binary encoding*) [15]. Все такие способы обеспечивают некоторое представление исходной переменной с ограниченным доменом в SAT-решателе в виде набора литералов, однако различаются возможностями их использования в различных контекстах: *onehot*-кодирование используется, когда необходимо получить доступ к значению переменной (здесь под «доступом» подразумевается использование переменной в декларативном процессе построения сведения); *order*-кодирование используется, если необходимо закодировать порядок между, например, целочисленными переменными; *binary*-кодирование позволяет эффективно кодировать арифметические операции в сведении; гибридное *onehot* + *binary*-кодирование обеспечивает комбинацию возможностей этих двух способов кодирования.

Описанные выше детали кодирования переменных с ограниченными доменами должны быть учтены при построении сведения, поэтому возникает желание автоматизировать эти действия. Разработанный фреймворк kotlin-satlib содержит отдельный модуль Vars, предназначенный для манипуляции такими переменными. Основой модуля является класс DomainVar<T>, хранящий набор литералов, соответствующих *onehot*-кодировке переменной со значениями произвольного типа T. При создании экземпляра такой переменной имеется возможность указать один из способов кодирования: *onehot* или *onehot* + *binary*. Класс DomainVar<T> также содержит инфиксные методы eq(value: T) и neq(value: T), позволяющие обращаться к литералу, соответствующему значению value. Например, пусть `v: DomainVar<Int>`, тогда простое и удобное в использовании выражение `v eq 5` соответствует литералу, кодирующему $v = 5$.

Модуль манипуляции массивами SAT переменных

Некоторые переменные сведения могут быть объявлены с множественными индексами, что подразумевает их хранение в многомерном массиве. Ввиду того, что стандартные многомерные массивы, доступные в Java (например, int[][][]) и Kotlin (например, Array<Array<Array<Int>>>) не обеспечивают необходимой гибкости и удобств, была разработана собственная эффективная реализация многомерных массивов, оформленная в виде отдельной библиотеки MultiArray[1] на языке Kotlin.

Библиотека MultiArray включает в себя набор интерфейсов и их реализаций для многомерных массивов, индексируемых с нуля или единицы (по выбору), хранящих либо значения произвольного типа, либо (в отдельных эффективных специализациях) целочисленные и булевы значения. Основные интерфейсы, предоставляемые библиотекой — изменяемые (MutableMultiArray<T>) и неизменяемые (MultiArray<out T>) многомерные массивы. Неизменяемая версия ковариантна по типу T, что позволяет обращаться с такими массивами более гибко. Внутри многомерных массивов данные хранятся в одном линейном массиве, а индексация происходит с помощью так называемых *strides of array* («шаги массива»). Такой подход позволяет получить максимальную эффективность в совокупности с удобством по сравнению со стандартными массивами, встроенными в языки Java и Kotlin.

[1] https://github.com/Lipen/MultiArray

Экспериментальное исследование

Произведен синтез минимальных формул для всех 256 булевых функций от $X = 3$ переменных с использованием двух подходов поиска минимального значения параметра P — размера дерева разбора синтезируемой формулы:

1) итеративный перебор с перезапуском SAT-решателя на каждом шаге (суммарное время — 171 с);
2) инкрементальное расширение сведения с использованием предположений на каждом шаге (суммарное время — 185 с).

Результаты проведенного экспериментального сравнения представлены на рисунке, *а*, где оси соответствуют времени (в секундах, в логарифмической шкале) поиска минимальной булевой формулы двумя подходами, а каждая точка (всего 256 точек) соответствует отдельной булевой функции от $X = 3$ переменных. Пунктирная линия (*baseline*) соответствует равенству времени работы двух подходов. Скопление точек сосредоточено около базовой линии, что свидетельствует о том, что оба подхода позволяют решать поставленную задачу примерно одинаково эффективно.

Можно заметить, что большинство минимальных формул для функций от трех переменных были найдены менее, чем за одну секунду — сравнение на таких масштабах времени в контексте решения NP-трудных задач не является целесообразным. В связи с этим был выполнен дополнительный набор экспериментов на данных большей размерности и более «сложных» булевых функциях — от $X = 5$ переменных. Результаты приведены на рисунке, *б*, где показаны только точки со временем работы, превышающим 10 с (всего 54 точки). Данный график — а именно, точки в правой части графика под базовой линией — позволяет судить о том, что инкрементальный подход действительно обеспечивает лучшую производительность в рассмотренной задаче синтеза минимальной булевой формулы.

Заключение

В работе рассмотрена NP-трудная задача синтеза минимальной булевой формулы по таблице истинности. Для ее решения использован подход программирования в ограничениях, а именно, сведение к задаче выполнимости SAT: предложена кодировка структуры дерева разбора синтезируемой формулы, а в качестве метрики оптимальности формулы использован размер дерева разбора — суммарное число операторов и терминалов в формуле.

Разработан фреймворк kotlin-satlib для удобной и эффективной программной реализации сведений NP-трудных задач к SAT. Фреймворк kotlin-satlib позволяет:

1) взаимодействовать с современными SAT-решателями (MiniSAT, Glucose, CaDiCaL, cryptominisat) через нативный интерфейс с помощью технологии JNI;
2) объявлять распространенные виды ограничений через предметно-ориентированный язык (DSL), включая поддержку преобразований Цейтина;
3) манипулировать SAT-переменными, а также их многомерными массивами – как логическими, так и переменными с ограниченным доменом (например, целочисленными переменными или с пользовательскими типами-перечислениями).

Экспериментальное исследование с применением разработанного фреймворка kotlin-satlib на примере задачи синтеза минимальной булевой формулы показало, что использование инкрементальных SAT-решателей для решения NP-трудных задач с помощью инкрементального подхода является целесообразным, однако ощутимый выигрыш в производительности по сравнению с неинкрементальными подходом наблюдается только на задачах большой размерности.

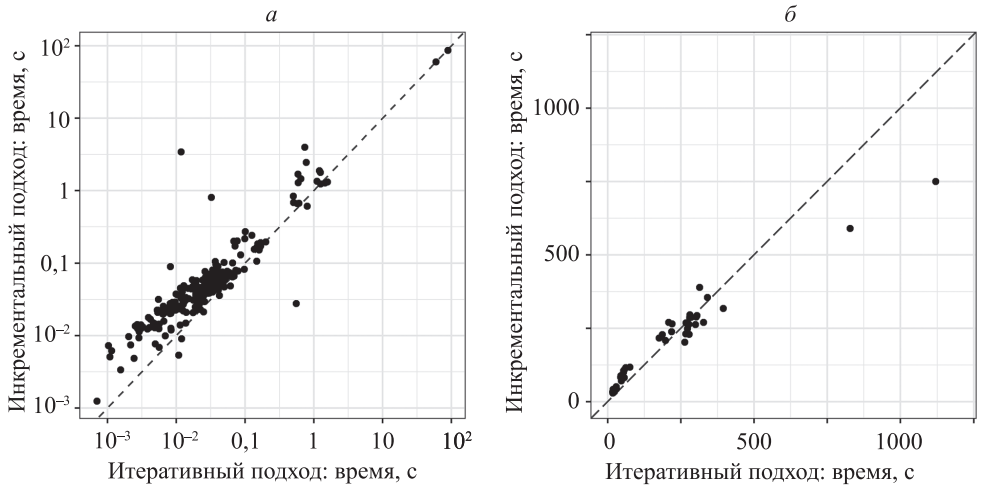

Рисунок. Графики сравнения времени работы алгоритма синтеза минимальной булевой формулы: от трех переменных (*а*); от пяти переменных (*б*) для двух подходов: итеративный (горизонтальная ось) и инкрементальный (вертикальная ось). Каждая точка на графике соответствует булевой функции

Литература

1. Akshay S., Chakraborty S., Goel S., Kulal S., Shah S. What's hard about boolean functional synthesis? // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2018. V. 10981. P. 251–269. doi: 10.1007/978-3-319-96145-3_14
2. Brayton R.K., Hachtel G.D., McMullen C.T., Sangiovanni-Vincentelli A.L. Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishers, 1984. 206 p. doi: 10.1007/978-1-4613-2821-6
3. Fišer P., Kubátová H. Flexible two-level boolean minimizer BOOM-II and its applications // Proc. 9th EUROMICRO Conference on Digital System Design: : Architectures, Methods and Tools (DSD 2006). Dubrovnik, Croatia. 2006. P. 369–376. doi: 10.1109/DSD.2006.53
4. Biere A., Heule M., van Maaren H., Walsh T. Handbook of Satisfiability. IOS Press, 2009. 980 p.
5. Cook S.A. The complexity of theorem-proving procedures // Proc. 3rd Annual ACM Symposium on Theory of Computing. 1971. P. 151–158. doi: 10.1145/800157.805047
6. Marques Silva J.P., Sakallah K.A. GRASP — A new search algorithm for satisfiability // Proc. IEEE/ACM International Conference on Computer-Aided Design. 1996. P. 220–227. doi: 10.1109/ICCAD.1996.569607
7. Ulyantsev V., Zakirzyanov I., Shalyto A. BFS-based symmetry breaking predicates for DFA identification // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2015. V. 8977. P. 611–622. doi: 10.1007/978-3-319-15579-1_48
8. Tseitin G.S. On the complexity of derivation in propositional calculus // Automation of Reasoning: 2: Classical Papers on Computational Logic. Berlin: Springer Berlin Heidelberg, 1983. P. 466–483. doi: 10.1007/978-3-642-81955-1_28
9. Biere A. PicoSAT Essentials // Journal on Satisfiability, Boolean Modeling and Computation. 2008. V. 4. N 2-4. P. 75–97. doi: 10.3233/sat190039
10. Eén N., Sörensson N. An extensible SAT-solver // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2004. V. 2919. P. 502–518. doi: 10.1007/978-3-540-24605-3_37
11. Audemard G., Simon L. Predicting learnt clauses quality in modern SAT solvers // Proc. 21st International Joint Conference on Artifical Intelligence (IJCAI 2009). 2009. P. 399–404.
12. Soos M., Nohl K., Castelluccia C. Extending SAT solvers to cryptographic problems // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2009. V. 5584. P. 244–257. doi: 10.1007/978-3-642-02777-2_24
13. Walsh T. SAT v CSP // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2000. V. 1894. P. 441–456. doi: 10.1007/3-540-45349-0_32
14. Nguyen V.-H., Mai S.T. A new method to encode the at-most-one constraint into SAT // ACM International Conference Proceeding Series. 2015. P. 46–53. doi: 10.1145/2833258.2833293
15. Björk M. Successful SAT encoding techniques // Journal on Satisfiability, Boolean Modeling and Computation. 2011. V. 7. N 4. P. 189–201. doi: 10.3233/SAT190085
16. Petke J., Jeavons P. The order encoding: From tractable CSP to tractable SAT // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2011. V. 6695. P. 371–372. doi: 10.1007/978-3-642-21581-0_34

References

1. Akshay S., Chakraborty S., Goel S., Kulal S., Shah S. What's hard about boolean functional synthesis? *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 10981, pp. 251–269. doi: 10.1007/978-3-319-96145-3_14
2. Brayton R.K., Hachtel G.D., McMullen C.T., Sangiovanni-Vincentelli A.L. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984, 206 p. doi: 10.1007/978-1-4613-2821-6
3. Fišer P., Kubátová H. Flexible two-level boolean minimizer BOOM-II and its applications. *Proc. 9th EUROMICRO Conference on Digital System Design: : Architectures, Methods and Tools (DSD 2006)*. Dubrovnik, Croatia, 2006, pp. 369–376. doi: 10.1109/DSD.2006.53
4. Biere A., Heule M., van Maaren H., Walsh T. *Handbook of Satisfiability*. IOS Press, 2009, 980 p.
5. Cook S.A. The complexity of theorem-proving procedures. *Proc. 3rd Annual ACM Symposium on Theory of Computing*, 1971, pp. 151–158. doi: 10.1145/800157.805047
6. Marques Silva J.P., Sakallah K.A. GRASP — A new search algorithm for satisfiability. *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1996, pp. 220–227. doi: 10.1109/ICCAD.1996.569607
7. Ulyantsev V., Zakirzyanov I., Shalyto A. BFS-based symmetry breaking predicates for DFA identification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015, vol. 8977, pp. 611–622. doi: 10.1007/978-3-319-15579-1_48
8. Tseitin G.S. On the complexity of derivation in propositional calculus. *Automation of Reasoning: 2: Classical Papers on Computational Logic*. Berlin, Springer Berlin Heidelberg, 1983, pp. 466–483. doi: 10.1007/978-3-642-81955-1_28
9. Biere A. PicoSAT Essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 2008, vol. 4, no. 2-4, pp. 75–97. doi: 10.3233/sat190039
10. Eén N., Sörensson N. An extensible SAT-solver. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2004, vol. 2919, pp. 502–518. doi: 10.1007/978-3-540-24605-3_37
11. Audemard G., Simon L. Predicting learnt clauses quality in modern SAT solvers. *Proc. 21st International Joint Conference on Artifical Intelligence (IJCAI 2009)*, 2009, pp. 399–404.
12. Soos M., Nohl K., Castelluccia C. Extending SAT solvers to cryptographic problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009, vol. 5584, pp. 244–257. doi: 10.1007/978-3-642-02777-2_24
13. Walsh T. SAT v CSP. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2000, vol. 1894, pp. 441–456. doi: 10.1007/3-540-45349-0_32
14. Nguyen V.-H., Mai S.T. A new method to encode the at-most-one constraint into SAT. *ACM International Conference Proceeding Series*, 2015, pp. 46–53. doi: 10.1145/2833258.2833293
15. Björk M. Successful SAT encoding techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 2011, vol. 7, no. 4, pp. 189–201. doi: 10.3233/SAT190085
16. Petke J., Jeavons P. The order encoding: From tractable CSP to tractable SAT. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6695, pp. 371–372. doi: 10.1007/978-3-642-21581-0_34

Авторы

Чухарев Константин Игоревич — программист, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, ORCID: 0000-0002-4636-2379, kchukharev@itmo.ru

Authors

Konstantin I. Chukharev — Software Developer, ITMO University, Saint Petersburg, 197101, Russian Federation, ORCID: 0000-0002-4636-2379, kchukharev@itmo.ru