УДК 512.552+519.95

©2013. **В. В. Скобелев**

ПРОБЛЕМА ПРОВЕРКИ ВЫПОЛНИМОСТИ ФОРМУЛ РАЗРЕШИМЫХ ТЕОРИЙ (ОБЗОР)

Данная работа посвящена анализу современного состояния исследований проблемы проверки выполнимости формул разрешимых теорий 1-го порядка на основе «ленивого подхода», т.е. на интеграции SAT-решателей с $\mathcal{T}$-решателями. Охарактеризована структура SAT-решателя, построенного на основе управляющей конфликтами DPLL-процедуре. Рассмотрены основные понятия и принципы, используемые в процессе построения современных $\mathcal{T}$-решателей. Изложение иллюстрируется на примере решателя, предназначенного для анализа выполнимости формул линейной целочисленной арифметики. Охарактеризованы методы организации взаимодействия SAT-решателей и $\mathcal{T}$-решателей.

Ключевые слова: *теории 1-го порядка, выполнимость формул, решатели.*

1. Введение. Применение информационных технологий практически во всех сферах деятельности человечества привело к необходимости автоматизации процессов управления, проектирования и сопровождения реальных (в том числе, информационных) систем с критической областью применения. Широкий класс реальных задач (организация потоков информации в компьютерных сетях, формальная верификация систем, представление и обработка знаний, планирование, исследование операций, тестирование дискретных устройств и т.д.), а также задач дискретной математики (основанных на использовании теоретико-множественных, теоретико-числовых, логических, графовых и сетевых моделей) имеет следующее общее свойство: решение задачи естественно сводится к *проверке выполнимости* некоторой *формулы* (см., напр., [1-10]). Именно это свойство и лежит в основе разработки средств автоматизированного решения (т.е. *решателей*) всех таких задач.

Замечание 1. Выполнимость формулы математической логики означает существование интерпретации (говорят также, модели), в которой она истинна (см., напр., [11, 12]).

Проверку выполнимости формулы F исчисления высказываний можно осуществить с помощью следующей рекурсивной схемы. Рассмотрим двоичное размеченное ориентированное ранжированное дерево $\mathfrak{D}_F$, построенное в соответствии со следующими правилами:

1. Корень дерева $\mathfrak{D}_F$ отмечен формулой F.

2. Осуществляется выбор висячей вершины v дерева $\mathfrak{D}_F$ и переменной x, входящей в формулу F_v, являющуюся отметкой вершины v.

3. Из вершины v выходят две дуги, ведущие в вершины v_l и v_r следующего уровня. Дуга, ведущая в вершину v_l (соответственно, в вершину v_r), имеет отметку $x \mapsto \top$ (соответственно, $x \mapsto \bot$). Отметка вершины v_l (соответственно, вершины v_r) – формула, полученная из F_v подстановкой $x \mapsto \top$ (соответственно, $x \mapsto \bot$), и упрощенная на основе тождеств исчисления высказываний (символы $\top$ и $\bot$ означают, соответственно, логические значения `true` и `false`).

Построение дерева $\mathfrak{D}_F$ осуществляется до тех пор, пока либо не появится вершина с отметкой $\top$, либо каждая висячая вершина имеет отметку $\bot$ (такое поддерево дерева $\mathfrak{D}_F$

называется *семантическим деревом* формулы F). В 1-м случае формула F – выполнимая, а во 2-м случае – невыполнимая (т.е. тождественно ложная во всех интерпретациях). Подчеркнем, что семантическое дерево (в отличие от таблиц истинности) осуществляет (вообще говоря, частичное) присвоение истинностных значений булевым переменным с целью поиска одной (безразлично, какой именно) интерпретации, в которой исследуемая формула исчисления высказываний является выполнимой.

Приведенная выше схема представляет класс рекурсивных алгоритмов, каждый из которых определяется правилами выбора раскрываемой вершины v и переменной x, входящей в формулу F_v. Известно, что (см., напр., [13]) проверка выполнимости формул исчисления высказываний – NP-полная задача.

Значительные усилия были затрачены на разработку методов, позволяющих упростить решение этой задачи на практике. Один из классов таких методов основан на применении *правила резолюции*: следствием дизъюнктов $A \vee B$ и $\overline{A} \vee C$ является дизъюнкт $B \vee C$ (этот дизъюнкт называется *резольвентой*). Известно, что формула невыполнима тогда и только тогда, когда конечным числом применений правила резолюции из нее выводится значение $\perp$. Комбинация построения семантического дерева и применения правила резолюции и лежит в основе DPLL-процедуры [14, 15] (DPLL – сокращение от Davis-Putnam-Logemann-Loveland).

Для исчисления предикатов ситуация иная. Известно, что не существует алгоритм проверки выполнимости формул исчисления предикатов. Однако существуют алгоритмы, которые для любой выполнимой формулы исчисления предикатов за конечное число шагов устанавливает ее выполнимость. Идея их построения состоит в следующем.

Замкнутая (т.е. не содержащая свободных переменных) формула F_1 преобразуется в эквивалентную формулу F_2, не содержащую логические связки, отличные от $\vee$, $\wedge$ и $^-$. Формула F_2 приводится к *предваренной нормальной форме* $F_4 = (Q_1 x_1) \ldots (Q_n x_n) F_3$, где $Q_1, \ldots, Q_n$ – кванторы, а F_3 – бескванторная формула. Элиминацией кванторов существования формула F_4 приводится к *сколемовской нормальной форме* $F_6 = (\forall y_1) \ldots (\forall y_m) F_5$, где F_5 – бескванторная формула. В силу теоремы Эрбрана невыполнимость формулы F_6 (т.е. используется метод доказательства «от противного») эквивалентна существованию конечного невыполнимого подмножества множества формул, полученных подстановкой в F_5 всевозможных термов, которые можно построить при помощи предметных констант и функциональных символов, встречающихся в F_5. Таким образом, проверка невыполнимости формулы F_6, по своей сути, сводится к проверке невыполнимости конечного множества формул исчисления высказываний.

Задача, представленная формулой математической логики, может быть решена применением SAT-решателя (т.е. программной системы, предназначенной для проверки выполнимости формул математической логики). Если задача представлена формулой разрешимой теории 1-го порядка $\mathcal{T}$, то говорят о выполнимости по модулю этой теории. В этом случае используется обозначение $SMT(\mathcal{T})$ (SMT – сокращение фразы «Satisfiability Modulo Theory»). Для решения таких задач применяется $\mathcal{T}$-решатель (т.е. программная система, осуществляющая анализ совместности ограничений, представленных атомами теории $\mathcal{T}$). Основы построения $\mathcal{T}$-решателей заложены в работах [4, 16-22].

В течение последнего десятилетия значительные усилия были направлены на исследование методов интеграции SAT-решателей и $\mathcal{T}$-решателей, что дает возмож-

ность «кодировать» задачу формулой математической логики и применить для анализа выполнимости последней SAT-решатель. При этом $\mathcal{T}$-решатели применяются только для проверки совместности (в теории $\mathcal{T}$) множеств литералов, кодирующих ограничения, представленные атомами теории $\mathcal{T}$. Такой подход (он получил имя «ленивый подход» – lazy approach) дает возможность существенно расширить класс решаемых задач. В настоящее время этот подход считается наиболее перспективным при разработке средств автоматизированного решения задач (см., напр., [23-32]).

Цель данного обзора – анализ современного состояния исследования проблемы $SMT(\mathcal{T})$ на основе «ленивого подхода». В п. 2 охарактеризованы SAT-решатели, основанные на использовании DPLL-процедуры. В п. 3 представлены основные понятия и принципы, лежащие в основе построения $\mathcal{T}$-решателей. В п. 4 рассмотрены методы организации взаимодействия SAT-решателей и $\mathcal{T}$-решателей. Заключение содержит ряд выводов.

2. SAT-решатели. Успехи в разработке методов повышения эффективности анализа выполнимости формул исчисления высказываний [33-37] привели к появлению ряда достаточно мощных SAT-решателей, основанных на использовании DPLL-процедуры (см., напр., [38-42]). Такие SAT-решатели, для краткости, называют DPLL. Существующие DPLL можно разбить на следующие два класса:

1. DPLL, основанные на методе ветвей и границ [43] (look-ahead DPLL [33]). В этих DPLL с каждой вершиной v дерева ассоциируется множество S_v литералов, которым еще не присвоено истинностное значение (напомним, что *литералом* называют переменную или ее отрицание). Каждый раз раскрывается вершина v, для которой множество S_v содержит наиболее перспективный литерал l. Раскрытие вершины состоит в присвоении истинностного значения $l \mapsto \top$, т.е. в построении вершины v' следующего уровня, в которую из v идет дуга с отметкой $l \mapsto \top$, и в ассоциировании с вешиной v' множества $S_{v'} = S_v \setminus \{l, \bar{l}\}$.

2. DPLL, управляющие конфликтами (conflict driven DPLL [36,37]). В этих DPLL реализован поиск с возвращением [43], основанный на анализе и устранении конфликтов, возникающих при каждом раскрытии вершины дерева, приводящем к невыполнимости анализируемой формулы.

Отметим, что в настоящее время не известны успешные применения DPLL, основанных на методе ветвей и границ, при исследований проблемы $SMT(\mathcal{T})$ на основе «ленивого подхода».

Замечание 2. Для проверки выполнимости формул исчисления высказываний DPLL, основанные на методе ветвей и границ, более эффективны (по затрачиваемому времени), чем DPLL, управляющие конфликтами. Таким образом, затраты, связанные с управлением конфликтами, являются, по-видимому, той ценой, которую приходится платить за возможность применения «ленивого подхода» к исследованию проблемы $SMT(\mathcal{T})$.

Охарактеризуем структуру DPLL, управляющих конфликтами.

С анализируемой формулой исчисления высказываний, представленной в виде КНФ, ассоциируется упорядоченная пара (φ, μ), где φ – множество дизъюнктов, входящих в КНФ, а μ – множество значений истинности, присвоенных переменным (первоначально, $\mu = \emptyset$). DPLL, управляющая конфликтами состоит из следующих

трех процедур.

1. *Препроцессорная обработка данных.* Эта процедура предназначена для упрощения анализируемого множества дизъюнктов φ. Основана на комбинации следующих методов [44-46]:

а) для каждой переменной x, встречающейся в множестве φ либо только без отрицания, либо только с отрицанием из множества φ удаляются все дизъюнкты, содержащие эту переменную (так как эти дизъюнкты являются выполнимыми), а в множество μ добавляется соответствующее истинностное значение этой переменной (а именно, $x \mapsto \top$, если переменная x входит в φ без отрицания, и $x \mapsto \bot$, если переменная x входит в φ с отрицанием);

б) для каждого дизъюнкта, принадлежащего множеству φ, удаляются все дизъюнкты, частью которых является этот дизъюнкт;

в) пары дизъюнктов, принадлежащих множеству φ, к которым применимо правило резолюции, заменяются их резольвентами, а истинностные значение переменных, по которым осуществляется свертка, добавляются в множество μ.

Замечание 3. После препроцессорной обработки данных исходная пара (φ, μ) преобразуется в экви-выполнимую пару (φ', μ'), т.е. формула $\varphi \wedge \mu$ выполнима тогда и только тогда, когда выполнима формула $\varphi' \wedge \mu'$.

2. *Ветвление.* Эта процедура реализует *прямой ход* поиска с возвращением и предназначена для присвоения истинностных значений литералам, встречающихся в обрабатываемом множестве дизъюнктов. Основана на использовании того или иного *эвристического метода* (или их комбинации). На практике используются следующие эвристические методы:

а) выбор литерала, наиболее часто встречающегося в дизъюнктах минимальной длины, и присвоение ему истинностного значения [3, 35];

б) выбор литерала, приводящего к минимальному (по мощности) анализируемому множеству дизъюнктов, и присвоение ему истинностного значения [47];

в) выбор литерала, в соответствии со списком приоритетов переменных, и присвоение ему истинностного значения [7, 48];

Замечание 4. Список приоритетов переменных составляется (с участием пользователя) при кодировании реальной задачи формулой математической логики, исходя из значения этих переменных для решаемой задачи.

г) выбор литерала из списка литералов, входящих в дизъюнкты, рассмотренные на предыдущем шаге, в соответствии со значением той или иной меры, оценивающей вклад литерала в построение решения задачи, и присвоение ему истинностного значения [38-40];

д) выбор литералов на основе дедукции [40, 46, 49], т.е. итеративном анализе, направленном на выделение множества ψ эквивалентных (для выполнимости анализируемой формулы) под-дизъюнктов, подстановке в обрабатываемое множество дизъюнктов новой пропозициональной переменной вместо элементов множества ψ, и присвоение этой переменной истинностного значения $\top$.

3. *Анализ конфликтов.* Эта процедура реализует *обратный ход* (backtracking) поиска с возвращением, предназначена для модификации множества значений ис-

тинности, присвоенных переменным, и состоит в следующем [34, 36, 42, 50].

Представим текущее состояние поиска с возвращением (см., напр., [43]) в виде пути

$$\begin{matrix} v_0 \\ \emptyset \end{matrix} \rightarrow \begin{matrix} v_1 \\ (l_1, D_1) \end{matrix} \rightarrow \begin{matrix} v_2 \\ (l_2, D_2) \end{matrix} \rightarrow \cdots \rightarrow \begin{matrix} v_n \\ (l_n, D_n) \end{matrix}, \tag{1}$$

идущего из корня v_0 дерева поиска в вершину v_n, где l_i $(i = 1, \ldots, n)$ – такой литерал, что присвоение значения истинности $l_i \mapsto \top$ порождает вершину v_i, а D_i – дизъюнкт, на основе которого присвоено это значение.

Анализ вершины v_n состоит в следующем. Проверяется, существует ли среди дизъюнктов, которые предстоит обработать, дизъюнкт, имеющий значение $\bot$.

Если такого дизъюнкта нет, то из вершины v_n продолжается прямой ход.

Если же существует такой дизъюнкт D, то следующим образом реализуется обратный ход (через $Res(D', D'')$ обозначена резольвента дизъюнктов D' и D''). Последовательно вычисляются резольвенты $R_1, R_2, \ldots$, где

$$R_i = \begin{cases} Res(D, D_n), & \text{если } i = 1 \\ Res(R_{i-1}, D_{n-i+1}), & \text{если } i \geq 2 \end{cases}.$$

Эти вычисления осуществляются до тех пор, пока не будет получена резольвента $R_j = \overline{l_r} \vee D'$, где $r \in \{j+1, \ldots, n\}$, а $D' = \bigvee\limits_{j=1}^{h} \overline{l_{i_j}}$ $(1 \leq i_1 \cdots < i_h \leq n - j)$.

Обратный ход состоит в том, что текущее состояние (1) поиска с возвращением заменяется текущим состоянием

$$\begin{matrix} v_0 \\ \emptyset \end{matrix} \rightarrow \begin{matrix} v_1 \\ (l_1, D_1) \end{matrix} \rightarrow \begin{matrix} v_2 \\ (l_2, D_2) \end{matrix} \rightarrow \cdots \rightarrow \begin{matrix} v_h \\ (l_h, D_h) \end{matrix} \rightarrow \begin{matrix} v_{h+1} \\ (\overline{l_r}, D) \end{matrix}.$$

После этого осуществляется анализ вершины v_{h+1}.

Выше охарактеризована структура DPLL, управляющих конфликтами. Отметим, что в соответствующих программных реализациях для сокращения временных и емкостных затрат используются также дополнительные приемы, в том числе и псевдослучайный выбор объектов (см., напр., [51]).

В последнее десятилетие значительные усилия были направлены на разработку формальных моделей DPLL и SMT-систем, построенных на основе «ленивого подхода». Такие модели определяются системой продукций (rule-based systems) (см., напр., [26, 52-54]), а их значение определяется следующими двумя факторами.

Во-первых, они являются частью математического аппарата, предназначенного для унифицированного построения решателей, построенных на основе «ленивого подхода».

Во вторых, появляется возможность доказать для разрабатываемых решателей такие основные, с позиции теории алгоритмов, свойства, как *корректность* (soundness), *полнота* и *остановка*.

Замечание 5. По-видимому, одной из первых таких формальных моделей является (построенная на основе модели, разработанной В. Аккерманом в середине XX века) *теория*

равенства и неинтерпретируемых функций $\mathcal{EUF}$ (equality and uninterpreted functions) [2, 55-57]. Эта теория является бесквантворной теорией 1-го порядка, определяемой обычными аксиомами равенства

$$x = x, \quad (x = y) \to (y = x), \quad (x = y) \wedge (y = z) \to (x = z)$$

и аксиомами конгруэтности (f – функциональный, а P – предикатный символы)

$$\bigwedge_{i=1}^{n} (x_i = y_i) \to (f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)),$$

$$\bigwedge_{i=1}^{n} (x_i = y_i) \to (P(x_1, \ldots, x_n) = P(y_1, \ldots, y_n)).$$

Существующие $\mathcal{EUF}$-решатели имеют полиномиальную сложность. Их внедрение на верхний уровень обработки замкнутых относительно конгруэнции структур данных является мощным средством выявления конфликтов.

3. $\mathcal{T}$-решатели. Построение современных $\mathcal{T}$-решателей осуществляется на основе следующего подхода, получившего имя *наслоение* (layering) [58, 59].

Выделяется такая последовательность подтеорий

$$\mathcal{T}_1 \subset \mathcal{T}_2 \subset \cdots \subset \mathcal{T}_n = \mathcal{T},$$

что проверка совместности ограничений для теории $\mathcal{T}_i$ $(i = 1, \ldots, n-1)$, проще, чем соответствующая проверка для теории $\mathcal{T}_{i+1}$. Конструируется последовательность $S_1, \ldots, S_n$ решателей возрастающей выразительности (и, как следствие, сложности), где S_i $(i = 1, \ldots, n)$ – $\mathcal{T}_i$-решатель. Если $\mathcal{T}_i$-решатель S_i установил, что исследуемое множество ограничений несовместно, то $\mathcal{T}$-решатель выдает `unsat` без обращения к решателям $S_{i+1}, \ldots, S_n$.

Кроме ответа `sat` (множество ограничений совместно) или `unknown` (вывод о совместности множества ограничений не сделан) предусмотрена возможность выдачи $\mathcal{T}$-решателем следствий, установленных в процессе анализа исследуемого множества ограничений, представленных атомами теории $\mathcal{T}$. В случае ответа `sat` эти следствия называются *леммами* теории $\mathcal{T}$. Аналогичным образом, кроме ответа `unsat` предусмотрена возможность выдачи $\mathcal{T}$-решателем *конфликтных множеств*, т.е. тех или иных подмножеств ограничений, наличие которых приводит к невыполнимости исследуемого множества ограничений.

Проилюстрируем особенности построения $\mathcal{T}$-решателей на примере задачи проверки выполнимости формул линейной арифметики $\mathcal{LA}$, т.е. атомы этой теории – это формулы вида

$$\sum_{i=1}^{n} a_i x_i \Diamond b \quad (\Diamond \in \{\leq, <, \neq, =, \geq, >\}).$$

Отметим, что эта задача наиболее полно проработана на текущий момент.

Пример. Для проверки выполнимости бесквантворных формул линейной рациональной арифметики $\mathcal{LA}(\mathbb{Q})$ разработан ряд эффективных на практике $\mathcal{LA}(\mathbb{Q})$-решателей, основанных на использовании симплекс-метода (см., напр., [2, 60-62]). Иная ситуация имеет место

в случае проверки выполнимости бескванторных формул линейной целочисленной арифметики $\mathcal{LA}(\mathbb{Z})$. Известно, что эта задача – NP-полная, а $\mathcal{LA}(\mathbb{Z})$-решатели, аналогичные предложенным в [63,64], далеко не всегда справляются с решением реальных задач. В [65] предложен следующий достаточно эффективный на практике $\mathcal{LA}(\mathbb{Z})$-решатель в предположении, что атомы имеют вид

$$\sum_{i=1}^{n} a_i x_i \Diamond b \quad (\Diamond \in \{=, \leq\}).$$

Вначале применяется $\mathcal{LA}(\mathbb{Q})$-решатель. Если он выдает `unsat`, то $\mathcal{LA}(\mathbb{Z})$-решатель также выдает `unsat` и прекращает работу. Если же $\mathcal{LA}(\mathbb{Q})$-решатель выдает `sat` (т.е. конфликты не обнаружены), то осуществляется проверка целочисленности значений переменных. В случае положительного ответа $\mathcal{LA}(\mathbb{Z})$-решатель выдает `sat` и прекращает работу. В случае отрицательного ответа активируется модуль, предназначенный для анализа системы линейных диофантовых уравнений.

Первая процедура проверяет, есть ли среди анализируемых уравнений такое уравнение $\sum\limits_i a_{hi} x_i + b_h = 0$, что НОД чисел a_{hi} не является делителем числа b_h. В случае положительного ответа $\mathcal{LA}(\mathbb{Z})$-решатель выдает `unsat` (так как анализируемая система линейных диофантовых уравнений несовместна) и прекращает работу. В случае отрицательного ответа активируется процедура, которая следующим образом последовательно преобразует каждое из анализируемых уравнений.

В уравнении

$$\sum_i a_{hi} x_i + b_h = 0 \tag{2}$$

выбирается наименьший по модулю ненулевой коэффициент a_{hk}. Возможны следующие два случая:

1. Пусть $|a_{hk}| = 1$. Тогда уравнение (2) приводится к виду $x_k = -\sum\limits_{i \neq k} \alpha_{kh} a_{hi} x_i - \alpha_{hk} b_h$, где $\alpha_{hk} = a_{hk} |a_{hk}|^{-1}$. Эта подстановка осуществляется во все остальные уравнения.

2. Пусть $|a_{hk}| > 1$. Тогда уравнение (2) приводится к виду

$$a_{hk}(x_k + \sum_{i \neq k} a_{hi}^{(q)} x_i + b_h^{(q)}) + \sum_{i \neq k} a_{hi}^{(r)} x_i + b_h^{(r)} = 0,$$

где $a_{hi}^{(q)}$ и $a_{hi}^{(r)}$ (соответственно, $b_h^{(q)}$ и $b_h^{(r)}$) – частное и остаток от деления a_{hi} на a_{hk} (соответственно, b_h на a_{hk}). Вводится новая переменная $x_t = x_k + \sum\limits_{i \neq k} a_{hi}^{(q)} x_i + b_h^{(q)}$. Эта подстановка осуществляется во все уравнения. Такое преобразование применяется к анализируемому уравнению (2) до тех пор, пока не возникнет 1-й случай (что всегда происходит через конечное число шагов).

Если при выполнении рассмотренной процедуры обнаружены конфликты, то $\mathcal{LA}(\mathbb{Z})$-решатель выдает `unsat` и прекращает работу. Если же конфликты не обнаружены, то полученная система $x_j = \sum\limits_{i \neq j} a_{ji} x_i + c_j$ линейных диофантовых уравнений используется в качесте подстановки в анализируемую систему неравенств и активируется модуль, предназначенный для анализа системы линейных неравенств.

Вначале каждое такое неравенство $\sum\limits_i a_i x_i + b \leq 0$, что НОД g чисел a_i не является делителем числа b преобразуется в неравенство $\sum\limits_i a_i g^{-1} x_i + \lceil b g^{-1} \rceil \leq 0$.

Затем активируется $\mathcal{LA}(\mathbb{Q})$-решатель. Если выдает `sat` , то осуществляется проверка целочисленности значений переменных. В случае положительного ответа $\mathcal{LA}(\mathbb{Z})$-решатель выдает `sat` и прекращает работу. В случае отрицательного ответа активируется модуль, реализующий метод ветвей и границ.

Этот модуль следующим образом рекурсивно разбивает анализируемую задачу на две подзадачи добавлением дополнительных ограничений в исследуемую формулу.

Пусть $\mathcal{LA}(\mathbb{Q})$-решатель ассоциировал с переменной x_k нецелочисленное значение α_k. Дополнительное ограничение для 1-й подзадачи – неравенство $x_k - \lfloor \alpha_k \rfloor \le 0$, а для 2-й подзадачи – неравенство $-x_k + \lceil \alpha_k \rceil \le 0$. После этого к каждой из подзадач применяется $\mathcal{LA}(\mathbb{Q})$-решатель. Эти вычисления осуществляются до тех пор, пока либо $\mathcal{LA}(\mathbb{Z})$-решатель выдает `sat`, либо будет установлено, что все подзадачи невыполнимы. В последнем случае $\mathcal{LA}(\mathbb{Z})$-решатель выдает `unsat`.

Исследования, связанные с разработкой процедур анализа формул $\mathcal{LA}$ естественно привели к выделению следующих двух подтеорий.

1. *Разностная логика* $\mathcal{DL}$ (difference logic) [52,66-68]. Атомы этой теории – формулы вида $x - y \Diamond a$, где $\Diamond \in \{\le, <, \ne, =, \ge, >\}$. Существующие решатели для $\mathcal{DL}(\mathbb{Q})$ и $\mathcal{DL}(\mathbb{Z})$ имеют полиномиальную сложность. Большинство из них основано на сведении проверки выполнимости множества неравенств к проверке отсутствия отрицательных циклов в графе ограничений (constraint graph) [69]. Последний определяется следующим образом. Вершины отмечены переменными. Из вершины с отметкой «x» идет дуга в вершину с отметкой «y» тогда и только тогда, когда анализируемое множество неравенств содержит неравенство $x - y \le a$. Эта дуга имеет отметку «a».

2. *Теория двух целых переменных на неравенство* $\mathcal{UTVPI}$ (unit-two-variable-per-inequality) [70,71]. Эта теория является подтеорией теории $\mathcal{LA}(\mathbb{Z})$, а ее атомы имеют вид $\pm x \pm y \le a$. Существующие $\mathcal{UTVPI}$-решатели имеют полиномиальную сложность. Большинство из них основано на итеративном построении транзитивного замыкания: при добавлении нового ограничения выводятся всевозможные следствия до тех пор, пока либо будет получено противоречие, либо будет достигнута неподвижная точка.

Анализ $\mathcal{EUF}$ и $\mathcal{LA}$ показал, что сложность решателей, во-многом, характеризуется такими свойствами теории $\mathcal{T}$, как «быть выпуклой» (convex) и «быть бесконечно устойчивой» (stably-infinite) [19]. Эти свойства определяются следующим образом.

Теория $\mathcal{T}$ *выпукла*, если для любой конъюнкции K ее литералов и для любой дизъюнкции $\bigvee\limits_{i=1}^{n}(x_i = y_i)$ (где x_i и y_i – переменные теории $\mathcal{T}$)

$$\left(K \models_{\mathcal{T}} \bigvee_{i=1}^{n}(x_i = y_i)\right) \Leftrightarrow (\exists i \in \mathbb{N}_n)(K \models_{\mathcal{T}} (x_i = y_i)).$$

Теория $\mathcal{T}$ *бесконечно устойчива*, если для любой $\mathcal{T}$-выполнимой формулы φ существует модель с бесконечной областью, в которой формула φ выполнима.

Отметим, что $\mathcal{EUF}$, $\mathcal{LA}(\mathbb{Q})$ и $\mathcal{DL}(\mathbb{Q})$ – бесконечно устойчивые и выпуклые теории, а $\mathcal{LA}(\mathbb{Z})$, $\mathcal{DL}(\mathbb{Z})$ и $\mathcal{UTVPI}$ – бесконечно устойчивые и не выпуклые теории.

Значительные усилия исследователей были направлены на построение решателей для работы с основными структурами данных. Рассмотрим их кратко.

Теория битовых векторов $\mathcal{BV}$ [55, 72-79] является теорией 1-го порядка с равенством. Она предназначена для анализа дискретных устройств, представленных на языке регистровых передач, а также для верификации программ. Ее основные операции – конкатенация, выбор подслова, сложение и умножение по заданному модулю, а также побитовые логические операции над векторами одной и той же длины. Эта теория не является ни выпуклой, ни бесконечно устойчивой. Известно, что проверка выполнимости безкванторных формул теории $\mathcal{BV}$ – NP-полная задача. Большинство существующих $\mathcal{BV}$-решателей, используя препроцессорные вычисления, кодируют исследуемую задачу в виде данных либо для SAT-решателя, либо для $\mathcal{LA}(\mathbb{Z})$-решателя.

Отметим, что выбор приемлемых на практике алгоритмов для $\mathcal{BV}$-решателя является одной из наиболее актуальных проблем в настоящее время.

Теория массивов $\mathcal{AR}$ [5, 8, 17, 55] является теорией 1-го порядка с равенством. Она предназначена для анализа поведения на уровне «массив/память» (что, в частности, актуально как при верификации программ, так и при организации тестирования программных систем). Сигнатура $\mathcal{AR}$ содержит два интерпретированных функциональных символа: $read$ и $write$ ($read(a, i)$ представляет элемент, записанный в массив a по адресу i, а $write(a, i, e)$ – результат записи элемента e в массив a по адресу i). Аксиомы $\mathcal{AR}$ имеют следующий вид

$$(\forall a)(\forall i)(\forall e)(read(write(a, i, e), i) = e),$$

$$(\forall a)(\forall i, j)(\forall e)((i \neq j) \rightarrow read(write(a, i, e), j) = read(a, j)),$$

$$(\forall a, b)((\forall i)(read(a, i) = read(b, i)) \rightarrow (a = b)).$$

Известно, что проверка выполнимости безкванторных формул теории $\mathcal{AR}$ – NP-полная задача.

Отметим, что существующие $\mathcal{AR}$-решатели применяются, как правило, в комбинации с решателями, предназначенными для других теорий.

Теория списков $\mathcal{LI}$ [55] представляет собой теорию 1-го порядка с равенством и является подтеорией *теории рекурсивных типов данных* $\mathcal{RDT}$ [5]. Сигнатура $\mathcal{LI}$ содержит три интерпретированных функциональных символа, представляющих основные операторы языка LISP: функцию $cons$, конструирующую в памяти объект, содержащий два значения или указатели на эти значения, и функции car и cdr, осуществляющие выбор, соответственно, 1-го и 2-го элементов объекта. Аксиомы $\mathcal{LI}$ имеют следующий вид

$$(\forall x)(cons(cad(x), cdr(x)) = x),$$

$$(\forall x, y)(car(cons(x, y)) = x), \quad (\forall x, y)(cdr(cons(x, y)) = y),$$

$$(\forall x)(f^n(x) \neq x) \quad (f \in \{car, cdr\}, n \in \mathbb{N}). \tag{3}$$

Замечание 6. Аксиомы (3) обеспечивают отсутствие «зацикливания» при формировании списков.

Известно, что проверка выполнимости бесквантворных формул $\mathcal{LI}$ осуществима за линейное время. Это обосновывает целесообразность применения существующих $\mathcal{LI}$-решателей при решении задач проверки корректности любых алгоритмов, использующих списки в качестве основных структур данных (в частности, при верификации программ и анализе алгоритмов обработки моделей с сетевой структурой).

4. Интеграция DPLL и $\mathcal{T}$-решателей. Обозначим через $\mathcal{T}$-DPLL систему, состоящую из взаимодействующих DPLL и $\mathcal{T}$-решателя, предназначенную для решения задачи $SMT(\mathcal{T})$ на основе «ленивого подхода». Входными данными для $\mathcal{T}$-DPLL системы является исследуемая $\mathcal{T}$-формула φ. Система автоматически конструирует (посредством кодировки атомов теории $\mathcal{T}$ пропозициональными переменными) пропозициональное представление $\varphi^{(p)}$ формулы φ. Все многообразие взаимодействий DPLL и $\mathcal{T}$-решателя в системе $\mathcal{T}$-DPLL естественно разбивается на следующие два класса.

1. *Off-line взаимодействие.* Функционирование $\mathcal{T}$-DPLL системы, основанной на off-line взаимодействии, осуществляется следующим образом.

Формула $\varphi^{(p)}$ подается на вход DPLL. Если установлено, что $\varphi^{(p)}$ – невыполнимая формула, то система $\mathcal{T}$-DPLL выдает `unsat` и останавливается. Если же построена модель $\mu^{(p)}$ формулы $\varphi^{(p)}$, то множество η атомов теории $\mathcal{T}$, построенных в соответствии с литералами, входящими в $\mu^{(p)}$, подается на вход $\mathcal{T}$-решателя. Если $\mathcal{T}$-решатель устанавливает, что множество атомов η выполнимо, то система $\mathcal{T}$-DPLL выдает `sat` и останавливается. Если же $\mathcal{T}$-решатель устанавливает, что множество атомов η невыполнимо, то выбирается конфликтное подмножество η_0 множества η. Множество $\overline{\eta_0}$, состоящее из отрицаний атомов, принадлежащих множеству η_0 кодируется дизъюнкцией D литералов, и формула $\varphi^{(p)} \wedge D$ подается на вход DPLL.

Замечание 7. Переход от пропозициональной формулы $\varphi^{(p)}$ к формуле $\varphi^{(p)} \wedge D$ называется построением *лемм по требованию* (lemmas on demand) [80].

Таким образом, DPLL рассматривается как «черный ящик» при off-line взаимодействии в системе $\mathcal{T}$-DPLL .

2. *On-line взаимодействие.* Такая $\mathcal{T}$-DPLL система представляет собой вариант DPLL, функционирующий как *перечислитель* (enumerator) моделей пропозициональной формулы, выполнимость интерпретаций в теории $\mathcal{T}$ которых проверяется $\mathcal{T}$-решателем. С исследуемой формулой φ такая $\mathcal{T}$-DPLL система ассоциирует множество $\mathcal{T}$-литералов μ, которым присвоены значения (первоначально, $\mu = \emptyset$).

Замечание 8. $\mathcal{T}$-литералом называется атом теории $\mathcal{T}$ или его отрицание.

Структура $\mathcal{T}$-DPLL системы, основанной на on-line взаимодействии, во многом, аналогична структуре DPLL, управляющей конфликтами. Основными являются следующие три процедуры.

1. *$\mathcal{T}$-препроцессорная обработка данных.* Эта процедура предназначена для упрощения формулы φ, а также для преобразования (если возникает необходимость) множества μ, сохраняющего $\mathcal{T}$-выполнимость формулы $\varphi \wedge \mu$. Большинство ее шагов является комбинацией шагов соответствующей процедуры для DPLL с определяемыми теорией $\mathcal{T}$ правилами вывода, применяемыми к $\mathcal{T}$-литералам формулы φ. Среди методов упрощения формулы φ выделяют *нормализацию* $\mathcal{T}$-атомов и *ста-*

тическое изучение (static learning) [58, 81-84].

Под *нормализацией* $\mathcal{T}$-атомов понимают их приведение к стандартному виду.

Замечание 9. В зависимости от теории $\mathcal{T}$ при *нормализации* могут применяться замена некоторых операций и отношений на двойственные, свойства ассоциативности, коммутативности, дистрибутивности, поглощения, та или иная сортировка, и т.д.

Статическое изучение состоит в проверке совместности рассматриваемого множества $\mathcal{T}$-атомов посредством анализа ограничений, определяемых простейшими эквивалентностями и конгруэнциями.

Если при выполнении $\mathcal{T}$-препроцессорной обработки данных обнаруживается конфликт, то $\mathcal{T}$-DPLL система выдает `unsat` и останавливается.

2. *$\mathcal{T}$-ветвление.* Эта процедура реализует *прямой ход* поиска с возвращением. Большинство ее шагов является комбинацией шагов соответствующей процедуры для DPLL с основанными на учете семантики теории $\mathcal{T}$ методами *раннего отсечения* (early pruning) и *$\mathcal{T}$-продвижения* ($\mathcal{T}$-propagating) [52, 54, 58, 81, 82, 85, 86].

Раннее отсечение предназначено для сужения пространства поиска при выборе литералов, которым присваиваются значения. Его суть состоит в том, что при обнаруженении $\mathcal{T}$-несовместности множества $\mathcal{T}$-литералов μ нет необходимости рассматривать никакое расширение множества μ. Реализация этого метода основана на том обстоятельстве, что построение современных $\mathcal{T}$-решателей на основе техники наслоения дает возможность эффективно использовать последовательность соответствующих $\mathcal{T}_i$-решателей. Кроме того, могут применяться такие эвристические методы, как отсечение множества μ при отсутствии его расширения на протяжении заданного числа шагов, или при присвоении в течение данного числа шагов значений только чисто пропозициональным литералам.

$\mathcal{T}$-продвижение состоит в следующем. При текущем вызове $\mathcal{T}$-решателя осуществляется попытка выводов вида $\eta \models_{\mathcal{T}} l$, где $\eta \subseteq \mu$, а l – $\mathcal{T}$-литерал, значение которому еще не присвоено. Если такие $\mathcal{T}$-литералы найдены, то они добавляются в множество μ.

3. *$\mathcal{T}$-анализ конфликтов.* Эта процедура реализует *обратный ход* поиска с возвращением. Большинство ее шагов является комбинацией шагов соответствующей процедуры для DPLL с основанными на учете семантики теории $\mathcal{T}$ методами *$\mathcal{T}$-возврата* ($\mathcal{T}$-backjumping) и *$\mathcal{T}$-изучения* ($\mathcal{T}$-learning) [25, 34, 59, 86-91].

$\mathcal{T}$-возврат основан на предположении о том, что если $\mathcal{T}$-решатель активируется на множестве $\mathcal{T}$-литералов μ, то при установлении $\mathcal{T}$-несовместности множества μ он строит конфликтное подмножество $\eta \subset \mu$. В этом случае система $\mathcal{T}$-DPLL использует пропозициональное представление $\eta^{(p)}$ в качестве источника конфликта. При этом активируется режим возврата DPLL (т.е. $\varphi^{(p)} := \varphi^{(p)} \wedge D$, где D – дизъюнкция отрицаний пропозициональных литералов, принадлежащих $\eta^{(p)}$ и осуществляется возврат в вершину дерева поиска (какую именно, зависит от выбранной стратегии), в которой не присвоены значения ни одному из литералов, принадлежащих множеству $\eta^{(p)}$).

$\mathcal{T}$-изучение предназначено для выделения суб-минимальных по мощности конфликтных множеств $\mathcal{T}$-литералов, объединение которых содержит как можно боль-

ше $\mathcal{T}$-литералов, значения которым не присвоены на текущий момент. Именно такие системы множеств $\mathcal{T}$-литералов, а также аналогичные системы множеств чисто пропозициональных литералов используются при построении стратегии ветвления и возврата $\mathcal{T}$-DPLL системы в вершины дерева поиска, расположенные в как можно ранее построенных уровнях, что, в конечном итоге, обеспечивают наиболее существенное сужение пространства поиска.

Замечание 10. Наиболее простой такой стратегией ветвления и возврата $\mathcal{T}$-DPLL системы является следующая (см. процедуру анализа конфликтов для DPLL). Пусть S – множество $\mathcal{T}$-литералов, которым на текущий момент присвоены значения $\top$. Для каждого множества η, принадлежащего построенной на текущий момент системе суб-минимальных по мощности конфликтных множеств $\mathcal{T}$-литералов проверяется условие $|S \cap \eta| = |\eta| - 1$. Если это условие выполнено, то $\mathcal{T}$-литералу $l \in \eta \backslash S$ автоматически присваивается значение $\bot$. После окончания этого процесса происходит активация процедуры $\mathcal{T}$-анализа конфликтов.

5. Заключение. В работе рассмотрены модели и методы, предназначенные для исследования проблемы $SMT(\mathcal{T})$ на основе «ленивого подхода».

В настоящее время достаточно хорошо проработаны теоретические основы построения $\mathcal{T}$-DPLL систем в терминах разрешимых теорий 1-го порядка. При этом созданы общие методы синтеза $\mathcal{T}$-DPLL систем на основе сведения предназначенных для различных теорий решателей в единый решатель с последующей его интеграцией с DPLL [2, 17, 21, 39, 92].

Построение приемлемых на практике $\mathcal{T}$-решателей дает возможность конструировать на основе наслоения такие программные системы с достаточно широкой областью применения, как MathSAT [58], представляющий собой взаимодействующую с DPLL иерархию решателей, предназначенных, для проверки выполнимости формул, соответственно, теорий $\mathcal{EUF}$, $\mathcal{DL}$, $\mathcal{LA}(\mathbb{Q})$ и $\mathcal{LA}(\mathbb{Z})$. Возможности таких программных систем могут быть существенно расширены за счет включения в иерархию решателей, предназначенных для проверки выполнимости формул других теорий. Например, предложенного в [93] решателя, основанного на интервальной арифметике и предназначенного для проверки выполнимости системы нелинейных ограничений (в том числе построенных с помощью трансцендентных функций).

Анализ ситуации, связанной с построением решателей, предназначенных для работы с основными структурами данных, приводит к следующим выводам.

Во-первых, возможность построения эффективных $\mathcal{LI}$-решателей делает привлекательным их использование в автоматизированных средствах анализа сетевых моделей, а также в автоматизированных средствах верификации программных систем.

Во-вторых, актуальность построения приемлемого на практике $\mathcal{BV}$-решателя обусловлена не только прикладными задачами, но и задачей проверки выполнимости формул в кольцах вычетов. Значимость последней задачи обусловлена многочисленными применениями колец вычетов в задачах преобразования и защиты информации.

Из сказанного естественно вытекает необходимость исследования задачи проверки выполнимости формул в конечных кольцах. В [94] разработана общая схема

решателя, предназначенного для проверки выполнимости формул $\mathcal{LA}$ над конечным ассоциативным кольцом с ненулевым умножением. Существенны отличием от $\mathcal{LA}(\mathbb{Q})$ и $\mathcal{LA}(\mathbb{Z})$ является необходимость отдельного рассмотрения шести классов колец, определяемых наличием либо отсутствием коммутативности умножения, а также наличием либо отсутствием единицы или (в некоммутативных кольцах) левых либо правых единиц. Детальная проработка этой схемы для колец вычетов и матричных колец над конечным полем определяет возможное направление исследований. Другое направление связано с разработкой математического аппарата, предназначенного для проверки выполнимости формул нелинейной арифметики над конечными кольцами.

1. *Castellini C., Guinchiglia E., Tacchella A.* SAT-based planning in complex domains: concurrency, constraints and nondeterminism // Artificial Intelligence. – 2003. – N 1-2. – P. 85-117.
2. *Detlefs D., Nelson G., Saxe J.* Simplify: a theorem prover for program-checking // Journal of the ACM. – 2005. – N 3. – P. 365-473.
3. *Jeroslow R.G., Wang J.* Solving propositional satisfiability problems // Annals of Mathematics and Artificial Intelligence. – 1990. – **1** (1-4). – P. 167-187.
4. *Lin F., Zhao Y.* ASSAT: computing answer sets of logic program by SAT solvers // Artificial Intelligence. – 2004. – N 1-2. – P. 115-137.
5. *Oppen D.C.* Reasoning about recursively defined data structures // Journal of the ACM. – 1980. – N 3. – P. 403-411.
6. *Stephan P., Brayton R., Sangiovanni-Vincentelli A.* Combinational test generation using satisfiability // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 1996. – N 12. – P. 1167-1176.
7. *Strichman O.* Tuning SAT checkers for bounded model checking // LNCS. – 2000. – Vol. 1885. – P. 480-494.
8. *Stump A., Dill D.L., Barret C.W. at all.* A decision procedure for an extensional theory of arrays // Proc. of LICS'01. – 2001. – P. 29-37.
9. *Velev M.N., Bryant R.E.* Exploating positive equality and partial non-consistency in the formal verification of pipelined microprocessors // Proc. of DAC'99. – 1999. – P. 397-401.
10. *Velev M.N., Bryant R.E.* Effective use of Boolean satisfiabiliry procedures in the formal verification of superscalar and VLIW microptocessors // Journal of Symbolic Computation. – 2003. – N 2. – P. 73-106.
11. *Кудрявцев В.Б., Гассанов Э.Э., Подколзин А.С.* Введение в теорию интеллектуальных систем. – М.: МАКС Пресс, 2006. – 208 с.
12. *Bradley A.R., Manna Z.* The calculus of computation. Desicion procedures with applications to verification. – Berlin-Heidelberg: Springer-Verlag, 2010. – 366 p.
13. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. – М.: Мир, 1982. – 416 с.
14. *Davis M., Putnam H.* A computing procedure for quantification theory // Journal of the ACM. – 1960. – N 3. – P. 201-215.
15. *Davis M., Logemann G., Loveland D.* A machine programm for theorem proving // Journal of the ACM. – 1962. – N 7. – P. 394-397.
16. *Armando A., Giunchiglia E.* Embedding complex decision problems inside an interactive theorem prover // Annals of Mathematics and Artificial Intelligence. – 1993. – N 3-4. – P. 475-502.
17. *Nelson G., Oppen D.C.* Simplification by cooperating decision procedures // ACM Transactions on Programming Languages and Systems. – 1979. – N 2. – P. 245-257.
18. *Nelson G., Oppen D.C.* Fast decision procedures based on congruence closure // Journal of the ACM. – 1980. – N 2. – P. 356-364.
19. *Oppen D.C.* Complexity, convexity and combinations of theories // Theoretical Computer Science. – 1980. – N 12. – P. 291-302.

20. *Shostak R.* An algorithm for reasoning about equality // Com. ACM. – 1978. – N 3. – 583-585.
21. *Shostak R.* A practical decision procedure for arithmetic with function symbols // Journal of the ACM. – 1979. – N 2. – P. 351-360.
22. *Shostak R.* Deciding combinations of theories // Journal of the ACM. – 1984. – N 1. – P. 1-12.
23. *de Moura L., Ruass H., Sorea M.* Lazy theorem proving for bounded model checking over infinite domains // LNCS. – 2002. – Vol. 2392. – P. 438-455.
24. *de Moura L., Duterte B., Shankar N.* A tutorial on satisfiability modulo theoris // LNCS. – 2007. – Vol. 4590. – P. 20-36.
25. *Flanagan C., Joshi R., Ou X., at all.* Theorem proving using lazy proof explication // LNCS. – 2003. – Vol. 2725. – P. 438-455.
26. *Nieuwenhuis R., Oliveras A., Tinelli C.* Solving SAT and SAT Modulo theories: from an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL($\mathcal{T}$) // Journal of the ACM. – 2006. – N 6. – P. 937-977.
27. *Nieuwenhuis R., Oliveras A.* On SAT Modulo theories and optimisation problems // LNCS. – 2006. – Vol. 4121. – P. 156-169.
28. *Ranise S., Tinelli C.* Satisfiability modulo theories // IEEE Intelligent Systems Magazine. – 2006. – N 6. – P. 71-81.
29. *Ranise S., Tinelli C.* Satisfiability modulo theories library (SMT-LIB). – 2006. – http://www.SMT-LIB.org.
30. *Ranise S., Tinelli C.* The SMT-LIB Standard: Version 1.2. Technical Report, Department of Computer Science, The Univ. of Iowa, 2006. – http://www.SMT-LIB.org.
31. *Sebasttiani R.* Integrating SAT solvers with Math reasoners: Foundations and basic algorithms. // Technical Report 0111-22, ITC-IRST, Trento, Italy, 2001. – http://sra.itc.it/tr/Seb01.pdf.
32. *Sebasttiani R.* Lazy satisfiability modulo theories // Journal on Satisfiability, Boolean Modeling and Computation. – 2007. – N 3. – P. 141-224.
33. *Chu M.L., Anbulagan.* Look-ahead versus look-back for satisfiability problems // LNCS. – 1997. – Vol. 1330. – P. 341-355.
34. *Bajardo R.J., Schrag R.C.* Using CSP look-back techniques to solve real-world SAT instances // Proc. of AAAI'97. – 1997. – P. 203-208.
35. *Hooker J.N., Vinay V.* Branching rules for satisfiability // Journal of Automated Reasoning. – 1995. – N 3. – P. 359-383.
36. *Marques-Silva J., Sakallah K.* GRASP: a search algorithm for satisfiability // Proc. of ICCAD'96. – 1996. – P. 220-227.
37. *Marques-Silva J., Sakallah K.* GRASP: a search algorithm for propositional satisfiability // IEEE Transactions on Computers. – 1999. – N 5. – P. 506-521.
38. *Een N., Sorenson N.* An extensible SAT-solver // LNCS. – 2004. – Vol. 2919. – P. 502-518.
39. *Goldberg E., Novikov Y.* BerkMin: a fast and robust SAT-solver // Proc. of DATE'02. – 2002. – P. 142.
40. *Moskewich M.,W., Madigan C.F., Zhang Y.Z. at all.* Chaff: engineering an efficient SAT solver // Proc. of DAC'01. – 2001. – P. 530-535.
41. *Zhang H.* SATO: an efficient propositional prover // Proc. of DAC'97. – 1997. – P. 272-275.
42. *Zhang L., Malic S.* The quest for efficient boolean satisfiability solvers // LNCS. – 2002. – Vol. 2404. – P. 17-36.
43. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы. Теория и практика. – М.: Мир, 1980. – 476 с.
44. *Bacchus F., Winter J.* Effective prepocessing with hyper-resolution and equality reduction // Proc. of the 6^{th} International Symposium on Theoty and Applications of Satisfiability Testing. – 2003. – http://www.cs.toronto.edu/∼fbacchus/Parers/BSAT2003.pdf
45. *Brafman R.* A simplifier for propositional formulae with many binary clauses // IEEE Transactions on Systems, Man and Cybernetics. – 2004. – N 1. – P. 52-59.
46. *Een N., Biere A.* Effective prepocessing in SAT through variable and clause elimination // Proc. of SAT'05. – 2005. – http://mimisat.se/dounloads/SatELite.pdf
47. *Chu M.L., Anbulagan.* Heuristics based on unit propagation for satisfiability problems // Proc. of

IJCAI'97. – 1997. – P. 366-371.
48. *Guinchiglia E., Massarotto A., Sebastiani R.* Act, and the rest will follow: exploiting determinism in planning of satisfiability // Proc. of AAAI'98. – 1998. – P. 948-953.
49. *Chu M.L.* Integrating equivalency reasoning into davis-putnam procedure // Proc. of AAAI'00. – 2000. – P. 291-296.
50. *Zhang L., Madigan C.F., Moskewicz M.W., al all.* Efficient conflict-driven learning in boolean satisfiability // Proc. of ICCAD'01. – 2001. – P. 279-285.
51. *Gomes C.P., Selman B., Kautz H.* Boosting combinational search through randomization. // Proc. of AAAI'98. – 1998. – P. 431-437.
52. *Nieuwenhuis R., Oliveras A.* DPLL($\mathcal{T}$) with exhaustive theory propagation and its application to difference logic // LNCS. – 2005. – Vol. 3576. – P. 321-334.
53. *Nieuwenhuis R., Oliveras A., Tinelli C.* Abstract DPLL and abstract DPLL modulo theories // LNCS. – 2005. – Vol. 3452. – P. 36-50.
54. *Tinelli C.* A DPLL-based calculus for ground satisfiability modulo theories // LNAI. – 2002. – Vol. 2424. – P. 308-319.
55. *Mamma Z., Zarba C.* Combining decision procerures // LNCS. – 2003. – Vol. 2787. – P. 453-468.
56. *Nieuwenhuis R., Oliveras A., Tinelli C.* Congruence closure with integer offsets // LNAI. – 2003. – Vol. 2850. – P. 381-422.
57. *Nieuwenhuis R., Oliveras A.* Proof-producing congruence closure // LNCS. – 2005. – Vol. 3467. – P. 453-468.
58. *Bozzano M., Bruttomesso R., Cimatti A., at all.* MathSAT: A tight integration of SAT and mathematical decision procedures // Journal of Automated Reasoning. – 2005. – N 1-3. – P. 265-293.
59. *Bozzano M., Bruttomesso R., Cimatti A., at all.* An icremental and layered procedure for the satisfiability of linear arithmetic logic // LNCS. – 2005. – Vol. 3440. – P. 317-333.
60. *Borning A., Mariott K., Stuckey P.* Solving linear arithmetic constraints for user interface applications // Proc. of UIST'97. – 1997. – P. 87-96.
61. *Berezin S., Ganesh V., Dill D.L.* An online proof-producing decision procedure for mixed-integer linear arithmetic // LNCS. – 2003. – Vol. 2619. – P. 521-536.
62. *Duterte B., de Moura L.* A fast linear-arithmetic solver for DPLL(T) // LNCS. – 2006. – Vol. 4144. – P. 81-94.
63. *Faure G., Nieuwenhuis R., Oliveras A., at all.* SAT modulo the theory of linear arithmetic: exact, inexact and commersial solvers // LNCS. – 2008. – Vol. 4996. – P. 77-90.
64. *Pugh W.* The omega test: a fast and practical integer programming algorithm for dependence analysis //Proc. of ACM/IEEE Conference on Supercomputing. – 1991. – P. 4-13.
65. *Griggio A.* A practical approach to satisfiability modulo linear arithmetic // Journal on Satisfiability, Boolean Modeling and Computation. – 2012. – N 8. – P. 1-27.
66. *Cotton S., Maler O.* Fast and flexible difference constraint propagation for DPLL(T) // LNCS. – 2006. – Vol. – 4121. – P. 170-183.
67. *Mahfoudh M., Miebert P., Asarin E, at all.* A satisfiability checker for difference logic // Proc. of SAT'02. – 2002. – P. 222-230.
68. *Wang C., Ivancic F., Canai M.K., at all.* Deciding separation logic formulae by SAT and incremental negative cycle elimination // LNCS. – 2005. – Vol. 3835. – P. 322-336.
69. *Cherkassky B.V., Goldgerg A.V.* Negative cycle detection algorithms // Mathematical Programming. – 1999. – N 2. – P. 277-311.
70. *Harvey W., Stuckey P.* A unit two variable per inequality integer constraint solver for constraint logic programming // Proc. of ICAPS'05. – 2005. – P. 71-80.
71. *Lahiri S.K., Musuvathi M.* An efficient decision prosedure for UTVPI constraints // LNCS. – 2005. – Vol. 3717. – P. 168-183.
72. *Barret C. W., Dill D.L., Levitt J.R.* A decision procedure for bit-vector arithmetic // Proc. of The 35^{th} Design Automation Conference. – 1998. – P. 522-527.
73. *Bjorner N., Pichora M.C.* Deciding fixed and non-fixed size bit vectors // LNCS. – 1998. – Vol. 1384. – P. 376-392.
74. *Bozzano M., Bruttomesso R., Cimatti A., at all.* Encoding RTL Constructs for MathSAT // Electr.

Notes Theor. Comput. Sci. – 2006. – N 2. – P. 3-14.
75. *Brinkmann R., Drechsler R.* RTL-datapath verification using integer linear programming // Proc. of DAC'02. – 2002. – P. 741-746.
76. *Cyrluk D., Moller M.O., Ruess H.* An efficient decision procedure for the theory of fixed-sized bit-vectors // LNCS. – 1997. – Vol. 1254. – P. 60-71.
77. *Ganesh V., Dill D.L.* A desicion procedure for bit-vectors and arrays // LNCS. – 2007. – Vol. 4590. – P. 519-531.
78. *Fallah F., Devadas S., Keutzer K.* Functional vector generation for HDL models using linear programming and 3-satisfiability // Proc. of DAC'98. – 1998. – P. 355-367.
79. *Kovasznai G., Frohlich A., Biere A.* On the complexity of fixed-size bit-vectors logics with binary encoded bit-width // Proc. of SMT'12. – P. 44-55.
80. *Barret C., Nieuwenhuis R., Oliveras A., at all.* Splitting on demand in SAT modulo theories // LNCS. – 2006. – Vol. 4246. – P. 512-526.
81. *Armando A., Castellini C., Guinchiglia E.* SAT-based procedures for temporal reasoning // LNCS. – 2000. – Vol. 1809. – P.97-108.
82. *Audermand G., Bertolli P., Cimatti A., at all.* A SAT based approach for solving formulae over Boolean and linear mathematical propositions // 2002. – LNCS. – Vol. 2392. – P. 195-210.
83. *Seshia S., Lahiri S., Bryant R.* A hybrid SAT-based decision procedure for separation logic with uninterpreted functions // Proc. of DAC'03. – 2003. – P. 425-430.
84. *Strichman O., Seshia S.A., Bryant R.E.* Deciding separation formulae with SAT // LNCS. – 2004. – Vol. 2404. – P. 209-222.
85. *Barret C., Dill D., Stump A.* Checking satisfiability of first-order formulae by incremental translation to SAT // LNCS. – 2002. – Vol. 2404. – P. 236-249.
86. *Ganzinger H., Hagen G., Nieuwenhus R., at all.* DPLL(T): Fast decision procedures // LNCS. – 2004. – Vol. 3114. – P. 175-188.
87. *Bozzano M., Bruttomesso R., Cimatti A., at all.* Efficient satisfiability modulo theories via delayed theory combination // LNCS. – 2005. – Vol. 3576. – P. 335-349.
88. *de Moura L., Ruess H.* Lemmas on demand for satisfiability solvers // Proc. of SAT'02. – 2002. – P. 244-251.
89. *Horrocks I., Patel-Schneider P.F.* Optimizing propositional modal satisfiability for description logic subsumption // LNAI. – 1998. – Vol. 1476. – P. 234-246.
90. *Stump A., Barret C.W., Dill D.L.* CVC: a cooperative validity checker // LNCS. – 2002. – Vol. 2404. – P. 500-504.
91. *Wang C., Gupta A., Ganai M.* Predicate learning and selective theory deduction for a difference logic solver // Proc. of DAC'06. – 2006. – P. 235-240.
92. *de Moura L., Bjorner N.* Model based theory combination // Proc. of the 5^{th} Workshop on SMT (SMT'07). – 2007. – http://www.lsi.upc.edu/oliveras/smt07/.
93. *Franzle M., Herde C., Tiege T., at all.* Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure // Journal of Satisfiability, Boolen Modeling and Computation. – 2007. – N 1. – P. 209-236.
94. *Skobelev V. V.* Satisfiability modulo linear arithmetic over a finite ring // Вісник Київського університету. Сер.: фіз.-мат. науки. – 2013. – Вип. 2. – С. 95-106.

V. V. Skobelev

Problem of checking for satisfiability of formulae of decidable theories (survey).

Given paper is devoted to analysis of the state of the art for investigations of the problem of checking for satisfiability of formulae in decidable first-order theories on the base of the lazy approach, i.e. on integration of SAT-solvers with $\mathcal{T}$-solvers. The structure of SAT-solver designed on the base of conflict driven DPLL procedure is characterized. Basic notions and principles applied in the process of elaboration of modern $\mathcal{T}$-solvers are considered. They are presented in detail for example of a solver intended for

checking of satisfiability for formulae of linear integer arithmetic. Methods of integration of SAT-solvers with $\mathcal{T}$-solvers are characterized.

Keywords: *first order theories, formulae satisfiability, solvers.*

В. В. Скобелєв

Проблема перевірки здійсненості формул розв'язних теорій (огляд).

Дану статтю присв'ячено аналізу сучасного стану досліджень проблеми перевірки здійсненості формул теорій 1-го порядку на основі «ледащого підходу», тобто на інтеграції SAT-вирішувачів з $\mathcal{T}$-вирішувачами. Охарактеризовано структуру SAT-вирішувача, який побудовано на основі керуючою конфліктами DPLL-процедури. Розглянуто основні поняття та принципи, які використуються при побудові сучасних $\mathcal{T}$-вирішувачів. Викладення ілюструється на прикладі вирішувача, який призначено для перевірки здійсненості формул лінійної арифметики цілих чисел. Охарактеризовано методи інтеграції SAT-вирішувачів з $\mathcal{T}$-вирішувачами.

Ключові слова: *теорії 1-го порядку, здійсненость формул, вирішувачі.*

Ин-т прикл. математики и механики НАН Украины, Донецк
vv_skobelev@iamm.ac.donetsk.ua

Получено 10.05.13