Научная статья
УДК 334+37
DOI 10.17150/2411-6262.2021.12(4).11

В.И. Мартьянов
Байкальский государственный университет,
г. Иркутск, Российская Федерация

NP-ТРУДНЫЕ ЗАДАЧИ: АВТОМАТИЧЕСКОЕ ДОКАЗАТЕЛЬСТВО ТЕОРЕМ И МАШИНЫ ТЬЮРИНГА

АННОТАЦИЯ. Предлагается использовать для решения NP-полных (трудных) задач модификацию методов удовлетворения ограничением [1, с. 54] (УО) включением автоматического доказательства теорем (АДТ), а программирования в ограничениях [1, с. 114] — генераций машин Тьюринга (МТ) [2, с. 222]. В настоящее время УО [1, с. 57] использует АДТ в усеченной форме (логическое программирование), а предлагается использовать метод инвариантных преобразований (МИП) [3, с. 572], который является полноценным АДТ. Кроме того, предлагается использовать методы УО для генерации МТ, решающих NP-трудные задачи, записанные на ленте МТ, что является расширением возможностей программирования в ограничениях [1, с. 118].

КЛЮЧЕВЫЕ СЛОВА. NP-трудные задачи, удовлетворение ограничениям, программирование в ограничениях, искусственный интеллект, автоматическое доказательства теорем.

ИНФОРМАЦИЯ О СТАТЬЕ. Дата поступления 15 ноября 2021 г.; дата принятия к печати 23 ноября 2021 г.; дата онлайн-размещения 30 декабря 2021 г.

Original article

V.I. Martyanov
Baikal State University,
Irkutsk, Russian Federation

NP-DIFFICULT TASKS: AUTOMATIC PROOF OF THEOREMS AND TURING'S MACHINE

ABSTRACT. It is offered to use for solving NP-complete (difficult) tasks a modification of methods for satisfying the constraint [1, p. 54] (CS) by including automatic proof of theorems (APT), and programming in constraints [1, p. 114] — generation of Turing's machine (TM) [2, p. 222]. Currently, CS [1, p. 57] uses AP in a truncated form (logical programming), and it is suggested using the method of invariant transformations (MIT) [3, p. 572], which is a full-fledged APT. In addition, it is offered to use CS methods to generate TM solving NP-difficult tasks recorded on the TM tape, which is an extension of programming capabilities in constraints. [1, с. 118].

KEYWORDS. NP-difficult tasks, constraint satisfaction, constraint programming, artificial intelligence, automatic proof of theorems.

ARTICLE INFO. Received November 15, 2021; accepted November 23, 2021; available online December 30, 2021.

1. *Удовлетворение ограничениям в логическом программировании и метод инвариантных преобразований АДТ.*

Одной из важных задач искусственного интеллекта (ИИ) является задача удовлетворения ограничениям (УО) (constraint satisfaction problem). Целесообразность расширения концепции, предложенной *P. van Hentenrick*'ом в работе [4, p. 73], обосновывается следующими соображениями.

Основным и принципиальным недостатком классического метода удовлетворения ограничениям в логическом программировании, описанного

P. van Hentenrick'ом в [4, p. 78], является слишком простая структура организации данных, что будет показано в настоящем разделе на примере задачи проектирования расписания занятий учебных заведений.

Перечислим коротко основные недостатки метода удовлетворения ограничениям в логическом программировании:

Недостаточная выразительность описательных средств, слишком простая структура организации данных.

Эффективная работа обеспечивается для проверки выполнимости только бинарных ограничений.

Технические решения реализации ***P. van Hentenrick****'ом в [4, p. 73] метода удовлетворения ограничениям в логическом программировании не работают эффективно на сложноорганизованных множествах значений* A_1, A_2,..., A_n *даже при использовании только бинарных ограничений.*

Для иллюстрации рассмотрим постановку задачи проектирования расписания занятий для учебных заведений (в несколько упрощенной форме) и ее переформулировку в рамках метода удовлетворения ограничениям в логическом программировании.

Основные множества:

$\boldsymbol{L} = \{l_1,..., l_k\}$ — совокупность преподавателей,

$\boldsymbol{O} = \{o_1,..., o_m\}$ — совокупность дисциплин,

$\boldsymbol{C} = \{c_1,..., c_n\}$ — контингент учащихся (подгруппы, группы, потоки).

Множество проектируемых занятий: $\boldsymbol{P} \subseteq \boldsymbol{L} \times \boldsymbol{O} \times \boldsymbol{C}$.

Аудиторный фонд: $\boldsymbol{A} = \{a_1,..., a_q\}$.

Время (совокупность пар по дням недели): $\boldsymbol{T} = \{t_1,..., t_s\}$.

Ресурс: $\boldsymbol{R} = \boldsymbol{A} \times \boldsymbol{T}$.

Расписание является однозначным (инъективным) отображением:

$$\boldsymbol{P} \rightarrow \boldsymbol{R}, \tag{1}$$

которое должно удовлетворять весьма многочисленным условиям (из которых будут указаны лишь самые очевидные или бесспорные):

Общие и индивидуальные требования на дневное расписание преподавателей (количество занятий в день, включая лекции, освобожденные дни, пары и др.).

Общие и индивидуальные требования на дневное расписание контингентов учащихся (количество занятий в день, включая лекции, освобожденные дни, пары и др.).

Общие и индивидуальные требования на недельное расписание преподавателей (количество занятых дней, общее число единичных занятий, окон, сбалансированность недели и др.).

Общие и индивидуальные требования на недельное расписание контингентов учащихся (количество занятых дней, единичные занятия и окна исключаются, кроме физической культуры, сбалансированность недели и др.)

При постановке данной задачи **(1)** в стиле метода удовлетворения ограничениям:

$\boldsymbol{P} = \{p_1, p_2, ..., p_u\}$

– совокупность переменных , т.е. каждое занятие p_i объявляется переменной, которой сопоставлялся элемент ресурса

$$\boldsymbol{A_1} = ... = \boldsymbol{A_u} = \boldsymbol{R} \tag{2}$$

Замечание 1*. Конечно, так определенные области значений* **(2)** *существенно отличаются от классического определения* A_1, A_2, ..., A_n *областей значений, но*

*поддерживать корректность работы процедурно для такого задания не столь и сложно. Более того, стратегия «**проверяй вперед**» для данного варианта практически вырождается и не требует каких-либо специальных усилий и затрат алгоритмического ресурса.*

Все основные трудности связаны с проверкой ограничений. Ограничения при такой постановке задачи **(1)** формулируются для проекций (частей) переменных (так как переменная из P — вектор из трех составляющих: ***преподаватель, предмет, группа студентов***).

Аналогично и для областей значения переменных, где каждый элемент ресурса R также вектор из двух составляющих: ***аудитории и времени проведения проектируемого занятия***.

Для классического метода удовлетворения ограничениям в логическом программировании, описанного ***P. van Hentenrick'***ом в [4, p. 78], найти приемлемые технические решения по эффективной проверки такого рода ограничений практически невозможно.

Другим важным ограничением является отсутствие средств создания иерархических (или сетевых) структур данных, что, кстати, очень хорошо реализуется в реляционных базах данных и диалектах ***SQL***. С нашей точки зрения большие перспективы имеют средства создания структур ***связности*** на показателях, т.е. переменных $x_1, ..., x_n$ (или, что эквивалентно, на областях значений $A_1, ..., A_n$ переменных $x_1, ..., x_n$). Более подробно это было рассмотрено в задаче анализа плоских изображений [5, с. 35].

Существует весьма много формальных логических исчислений. С нашей точки зрения, главное их отличие — это та или иная степень возможности эффективной алгоритмизации для создания систем АДТ, а еще лучше — опыт эффективного решения прикладных задач большой размерности, как для рассматриваемого в данном разделе подхода на основе алгоритмизации элементарных шагов содержательных математических доказательств.

Формальные логические исчисления, ориентированные на программную реализацию, называются *системами автоматического доказательства теорем (АДТ)*. Полные системы автоматического доказательства теорем (например, метод резолюций) в принципе способны доказать любую теорему (тождественно истинную формулу), но не имеют возможности проверить ложность формулы (программа работает бесконечно и не останавливается).

Как уже упоминалось выше, при автоматизации процесса доказательства всех формул узкого исчисления предикатов (*УИП*) приходится сталкиваться с двумя фундаментальными проблемами, которые ограничивают возможности систем *АДТ*, основанных на алгоритмизации формальных систем логического вывода, в принципе:

При выводе сложно найти искомую цепочку вывода теоремы, так как возникает очень много комбинаций — происходит так называемый *комбинаторный взрыв*.

Если Φ — ложна, то это невозможно проверить, система «зацикливается».

Отметим еще раз, что для классической системы Гильберта [6, с. 75] построение цепочки доказательства, как правило, является практически неразрешимой задачей из-за комбинаторного взрыва, возникающего при построении цепочки доказательства.

Другие системы автоматического доказательства для некоторых классов формул строят цепочки более приемлемыми методами (например, метод резолюций [7, с. 196]).

В данном разделе будет рассмотрен эвристический метод АДТ, основанный на алгоритмизации элементарных шагов доказательств теорем содержательной математики.

Этот метод, так называемых инвариантных преобразований формул УИП, рассмотренный в работах [3, с. 578], и является предметом рассмотрения данного раздела (хотя рассмотрение будет проведено без углубления в детали обоснования полноты и области корректной работы данного метода, которые подробно изложены в работе [8, с. 79]).

Для содержательной математики характерно представление доказываемого утверждения в виде: *«Дано»*, *«Доказать»*, а так же *совокупности Σ посылок доказательства*, состоящей из аксиом, определений и ранее доказанных утверждений, рассматриваемого раздела математики.

Согласно этому, представим доказываемое утверждение в виде:

$$\Phi = \begin{cases} Дано: \alpha_1, \alpha_2, ..., \alpha_k \\ Доказать: \beta_1, \beta_2, ..., \beta_m \end{cases} \quad (3)$$

где α_i, β_i — некоторые утверждения.

Совокупность Σ посылок доказательства будем считать состоящей из выражений двух видов:

$$\alpha \supset \beta, \quad (4)$$

которые будем называть *продукциями* (по аналогии с терминологией, используемой в описаниях баз знаний экспертных систем и систем искусственного интеллекта);

$$\beta \equiv (\gamma_1 \supset \gamma_2), \quad (5)$$

которые будем называть *определениями.*

Пример 1. *Доказываемое утверждение* ***Ф*** **(3)** *и совокупность посылок доказательства (1) и (2):*

(0): *Ф = (A ⊆ B) & (B ⊆ C) ⊃ (A ⊆ C) — доказываемое утверждение, где «Дано»: (A ⊆ B), (B ⊆ C) «Доказать»: (A ⊆ C).*

(1): *(a ∈ B) & (B ⊆ C) ⊃ (a ∈ C) — продукция;*

(2): *(B ⊆ C) ≡ (a ∈ B) ∈ (a ∈ C) — определение.*

Замечание 2. *1) Применение продукций* **(4)** *к доказываемому утверждению* ***Ф*** *приводит к расширению «Дано» добавлением в него* ***β*** *(конечно, только в случае вхождения* ***α*** *в число утверждений «Дано»).*

Следовательно, осуществляется переход от доказательства утверждения ***Ф*** *к доказательству утверждения*

$$\Phi_1 = \begin{cases} Дано: \alpha_1, \alpha_2, ..., \alpha_k, \beta_* \\ Доказать: \beta_1, \beta_2, ..., \beta_m \end{cases}$$

2) Применение определений **(5)** *к доказываемому утверждению* ***Ф*** *приводит к переформированию «Доказать» заменой* ***β*** *на* γ_2 *и добавлением в «Дано» утверждения* γ_1 *(конечно, только в случае вхождения* ***β*** *в число утверждений «Доказать»).*

Следовательно, если предположить, что ***β*** *совпадает с* β_1*, то осуществляется переход от доказательства утверждения* ***Ф*** *к доказательству утверждения*

$$\Phi_2 = \begin{cases} Дано: \alpha_1, \alpha_2, ..., \alpha_k, \gamma_1 \\ Доказать: \gamma_2, \beta_2, ..., \beta_m \end{cases}$$

Рассмотрим доказательство утверждения ***Ф*** из примера ***1*** преобразованиями, неформально заданными в замечании ***2***.

Первым шагом будет применение преобразования, соответствующего *определению* (2) из примера ***1***, что приведет к переходу от доказательства утверждения Φ к доказательству утверждения Φ_1 равного

$$(A \subseteq B) \,\&\, (B \subseteq C) \,\&\, (a \in A) \supset (a \in C).$$

Вторым и третьим шагами будет применение преобразования, соответствующего *продукции* (1) из примера ***1***, что приведет к переходу от доказательства утверждения Φ_1 к доказательству утверждения Φ_2 равного

$$(A \subseteq B) \,\&\, (B \subseteq C) \,\&\, (a \in A) \,\&\, (a \in B) \,\&\, (a \in C) \supset (a \in C). \quad (6)$$

Нетрудно заметить, что утверждение Φ_2 удовлетворяет знаменитой формулировке из «Начал» Евклида «*получили, что и требовалось доказать*» (для этого термина даже было введено специальное обозначение в «Началах» Евклида).

В методе инвариантных преобразований (МИП) в качестве системы *аксиом* используются утверждения, соответствующие «*получили, что и требовалось доказать*», т.е. утверждения в которых, как в утверждении **(6)**, *«Дано» содержит «Доказать»*.

Изложим МИП более формально.

1. Доказываемые утверждения Φ — формулы УИП вида

$$\alpha_1 \,\&\, \alpha_2 \,\&\, \ldots \,\&\, \alpha_k \supset \beta_1 \,\&\, \beta_2 \,\&\, \ldots \,\&\, \beta_m \quad (7)$$

2. Преобразование, соответствующее *продукции* $\alpha \supset \beta$ **(1.3.2)**, преобразует формулу Φ в формулу Φ_1

$$\alpha_1 \,\&\, \alpha_2 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \beta \supset \beta_1 \,\&\, \beta_2 \,\&\, \ldots \,\&\, \beta_m \quad (8)$$

если α входит в число утверждений «Дано». Для всюду определенности применения продукций, как преобразования формул УИП, будем считать, что, если α *не входит* в число утверждений «Дано», то $\Phi_1 = \Phi$.

Таким образом, каждая *продукция* $\alpha \supset \beta$ (4) определяет преобразование $\varsigma\,[\alpha \supset \beta]$ всей совокупности формул УИП L_σ сигнатуры σ, т.е.

$$\varsigma\,[\alpha \supset \beta] : L_\sigma \to L_\sigma \quad (9)$$

Основной проблемой является *корректность* таким образом **(9)**, заданных ς — *преобразований*, т.е. совпадение логических значений формул Φ и $\varsigma\,[\alpha \supset = \supset \beta]\,(\Phi)$ для всех Φ из L_σ.

3. Преобразование, соответствующее *определению* $\beta \equiv (\gamma_1 \supset \gamma_2)$, **(9)**, преобразует формулу Φ в формулу Φ_1

$$(\alpha_1 \,\&\, \alpha_2 \,\&\, \ldots \,\&\, \alpha_k \,\&\, \gamma_1) \supset (\gamma_2 \,\&\, \beta_2 \,\&\, \ldots \,\&\, \beta_m) \quad (10)$$

если β входит в число утверждений «Доказать» (здесь β совпадает с β_1). Для всюду определенности применения продукций, как преобразования формул УИП, будем считать, что, если β *не входит* в число утверждений «Доказать», то $\Phi_1 = \Phi$.

Таким образом, каждая *определение* $\beta \equiv (\gamma_1 \supset \gamma_2)$ **(10)** определяет преобразование $\eta\,[\beta \equiv (\gamma_1 \supset \gamma_2)]$ всей совокупности формул УИП L_σ сигнатуры σ, т.е.

$$\eta\,[\beta \equiv (\gamma_1 \supset \gamma_2)] : L_\sigma \to L_\sigma \quad (11)$$

Основной проблемой является *корректность* таким образом **(11)**, заданных ***η*** *— преобразований,* т.е. совпадение логических значений формул ***Φ*** и ***η [β*** ≡ ≡ ***(γ₁ ⊃ γ₂)] (Φ)*** для всех ***Φ*** из L_{σ}.

Определение вида **(10)** дает переформулировку цели доказательства.

Допустим, что $\beta_1 \equiv (\gamma_1 \supset \gamma_2)$ (по определению), тогда в доказательстве утверждения Φ переходим к доказательству утверждения Φ_2:

$$\Phi_2 = \begin{cases} Дано: \alpha_1, \alpha_2, ..., \alpha_k, \gamma_1 \\ Доказать: \gamma_2, \beta_2, ..., \beta_m \end{cases}$$

Чаще всего выделяется ***лемма***, доказывающая утверждения вида:

$$\Phi_2^{'} = \begin{cases} Дано: \alpha_1, \alpha_2, ..., \alpha_k, \gamma_1 \\ Доказать: \gamma_2 \end{cases}$$

Если лемма будет доказана, то переходим к утверждению вида:

$$\Phi_2 = \begin{cases} Дано: \alpha_1, \alpha_2, ..., \alpha_k \\ Доказать: \beta_2, ..., \beta_m \end{cases}.$$

4. Как отмечалось выше, в методе инвариантных преобразований (МИП) в качестве системы *аксиом* используются утверждения, соответствующие «*получили, что и требовалось доказать*», т.е. утверждения в которых, как в утверждении **(11)**, *«Дано» содержит «Доказать»*. Таким образом, система аксиом МИП состоит из формул УИП, которые с некоторым упрощением, имеют вид

A & B ⊃ B **(12)**

Формулы УИП данной формы **(12)** будем называть ϖ — *каноническими.* В реализации МИП (системе АДТ АВТОДОТ) проверка на ϖ — *каноничность* производится процедурно.

5. Применение схемы доказательства методом от противного в МИП реализуется, как переход от доказательства формулы ***Φ*** вида ***A ⊃ B*** к доказательству формулы ***Φ₁*** вида

$$A \;\&\; \neg B \supset B \qquad (13)$$

Логическая эквивалентность формул ***Φ*** и ***Φ₁*** достаточно очевидна и, следовательно, переход от доказательства формулы ***Φ*** к доказательству формулы ***Φ₁*** **(13)** корректен.

Замечание 3*. При применении схемы доказательства методом от противного в МИП, в «Дано» на каком-то этапе, как правило, образуется противоречие, т.е. некоторое утверждение* ***a*** *и его отрицание* ***¬ a****. В реализации МИП (системе АДТ АВТОДОТ) проверка на присутствие такого рода противоречий производится процедурно (автоматическая проверка наличия противоречий включается после применения метода от противного).*

6. Применение схемы доказательства методом индукции в МИП реализуется, как переход от доказательства формулы ***Φ(x)*** вида ***A(x) ⊃ B(x)*** к доказательству формулы ***Φ₁*** вида

$$A(x) \;\&\; (A(y) \;\&(x > y) \supset B(y)) \supset B(x) \qquad (14)$$

Логическая эквивалентность формул ***Φ*** и ***Φ₁*** доказана в статье [3, с.576] и, следовательно, переход от доказательства формулы ***Φ*** к доказательству формулы ***Φ₁*** **(14)** корректен.

Естественным и необходимым требованием является определение переменной x на вполне упорядоченном множестве.

Замечание 4. 1) *При применении данной схемы реализации индукции не требуется доказательство выполнимости основания индукции.*

2) Естественно вместо порядка может быть использовано любое другое бинарное отношение, такое, что переменная x *определена на вполне упорядоченном множестве относительно этого отношения (иногда это может быть отношение делимости на целых числах, отношение включения подмножеств и т.д.).*

Реализация МИП система АДТ АВТОДОТ [3, с. 578] была успешно применена для решения двух прикладных задач сетевого планирования:

1) Формирование сетевого графика организационно-технических мероприятий обеспечения жизнеспособности территориально распределенной иерархической системы [9, с. 161].

2) Планирование информационных потоков в территориально распределенной иерархической системе [10, с. 46].

Для обеих задач применение МИП позволило получать совокупность планов в формуле вида:

$$A \,\&\, (B^1_1 \vee B^1_2) \,\&\ldots\&\, (B^k_1 \vee B^k_2) \supset C \qquad (15)$$

где допустимые планы (удовлетворяющие системно-техническим условиям), представлены некоторыми конъюнкциями в посылке импликации формулы **(15)**, получающимися после раскрытия дизъюнкций по правилу дистрибутивности, связывающих логические связки конъюнкции и дизъюнкции (т.е. привести посылку импликации формулы **(15)** к ***ДНФ***).

Понятно, что посылка импликации в форме ***ДНФ*** имеет 2 в степени k конъюнкций и поиск среди них, удовлетворяющих ограничениям (включая ϖ — *каноничность*), весьма непростая задача. Собственно говоря, эвристики, наработанные при решении данных задач, и стали частью техники решения *NP — трудных задач*, излагаемой в данной работе.

2. Программирование в ограничениях и генерация МТ.

Программирование в ограничениях (или *программирование ограничениями*) — парадигма программирования, в которой отношения между переменными указаны в форме ограничений. Ограничения отличаются от общих примитивов языков императивного программирования тем, что они определяют не последовательность шагов для исполнения, а свойства искомого решения, что делает такое программирование формой декларативного программирования.

Программирование в ограничениях тесно связано с теорией удовлетворения ограничений, которая предлагает удобный аппарат и простую формальную схему для представления и решения комбинаторных задач искусственного интеллекта. Исторически первая форма — логическое программирование с ограничениями, что уже рассмотрено в первой части данной статьи.

Во второй части этой статьи схематично (на уровне идеологии) рассмотрим генерацию МТ для решения NP — трудных задач, причем перебор программ МТ будем оптимизировать методами УО.

Машина Тьюринга имеет следующие части:

Конечная лента L, разделенная на ячейки. В процессе работы *ДМТ* с обеих сторон ленты могут пристраиваться новые ячейки, поэтому ленту L можно считать *потенциально бесконечной*. В каждой из ячеек прописан некоторый символ из *алфавита* $A = \{a_0, a_1, a_2, \ldots, a_n\}$, т.е. в ячейках ленты a_{ij} являются символами алфавита A. В процессе работы *МТ* символы в ячейках ленты L могут меняться

(или не меняться). Ленту L называют также *внешней памятью* MT. При пристраивании новой ячейки слева или справа предполагаем, что в ней прописывается один и тот же выделенный символ *алфавита* $A = \{a_0, a_1, ..., a_n\}$, а именно, a_0, причем в этом случае ячейка считается пустой.

Множество состояний MT $Q = \{q_0, q_1, q_2, ..., q_m\}$, элементы которого называются *внутренней памятью*. В дальнейшем предполагаем, что MT *всегда находится только в одном состоянии* (почему называется *детерминированной*, а *не детерминированная машина Тьюринга* может одновременно находиться более чем в одном состоянии). Состояние q_0 называется заключительным (или стоп-состоянием) при получении которого работа MT *прекращается*.

Управляющая головка. Это устройство считывает информацию *только* из одной ячейки ленты L.

Схема работы MT. Один *такт* работы MT состоит в считывания головкой информации из ячейки ленты MT и прописывания в эту ячейку некоторого символа *алфавита* $A = \{a_0, a_1, ..., a_n\}$ и сдвига головки (*обязательно!*) MT в соседнюю левую или правую ячейку, а $ДМТ$ *переходит* в некоторое состояние из множества $Q = \{q_0, q_1, q_2, ..., q_m\}$. Отметим, что МТ может работать *бесконечно* (или *вечно*), так как заключительное состояние q_0 может оказаться *недостижимым*. *Состоянием* (*машинным словом* или *конфигурацией*) МТ будем называть совокупность, образованную последовательностью $a_{j1}, a_{j2}, ...,$ символов всех ячеек ленты, состоянием внутренней памяти q_i и номером k воспринимаемой ячейки a_{jk}. Таким образом, каждое машинное слово содержит только ***одно*** вхождение символа состояния q_i. Для рассматриваемого здесь машинного слова символ q_i стоит на позиции k, что позволяет записать его следующей последовательностью

$$a_{j1}\, a_{j2} \ldots a_{j\,k-1}\, q_i\, a_{i\,k} \ldots a_{js} \qquad (16)$$

Важно отметить, что каждое машинное слово всегда заканчивается символом алфавита (если головка сдвигается вправо от последнего символа машинного слова, то справа приписывается символ a_0, т.е. справа к ленте приписывается пустая ячейка), но может начинаться символом состояния q_i.

Функции перехода, печати и сдвига. Функция перехода состояний

Pass : (Q x A) –> Q

Функция печати

Print : (Q x A) –> A

Функция сдвига головки

Displ : (Q x A) –> {–1, 1}

Рассмотрим преобразование машинного слова (16) после применения выше определенных функций. Пусть $Pass\,(q_i\,, a_{i\,k}) = q_\xi$; $Print\,(q_i, a_{i\,k}) = a_\lambda$; $Displ\,(q_i\,, a_{i\,k}) = t$. Тогда в зависимости от значения t (-1 соответствует сдвигу головки по ленте влево, а 1 — сдвигу головки вправо) возможны следующие два варианта.

1 вариант. Пусть $t = -1$. Тогда машинное слово (16) будет преобразовано в последовательность

$$a_{j1}\, a_{j2} \ldots q_\xi\, a_{j\,k-1}\, a_\lambda \ldots a_{js}.$$

2 вариант. Пусть $t = 1$. Тогда машинное слово (16) будет преобразовано в последовательность

$$a_{j1}\, a_{j2} \ldots a_{j\,k-1}\, a_\lambda\, q_\xi\, a_{j\,k+1} \ldots a_{js}.$$

Функции ***Pass, Print, Displ*** называют ***программой*** МТ, а начальную ленту L — ***задачей***. На ленте MT можно записывать классические NP — трудные задачи: выполнимости (SAT), раскраски графов, существования покрытия и др. [1, с. 114].

Методы *УО* можно применять для ускорения перебора входных данных (задачи на *ленте МТ*), а также для ускорения перебора *программ МТ (*функций ***Pass, Print, Displ****)*, что, возможно, обещает весьма неожиданные результаты.

Отметим, что *генерация программ* (хотя бы только для *МТ)* с нашей точки зрения является *принципиально новым* шагом, хотя пока мы не имеем по этому направлению весомых результатов.

Список использованной литературы

1. Щербина О.А. Удовлетворение ограничений и программирование в ограничениях / О.А. Щербина // Интеллектуальные системы. — 2011. — Т. 15, вып. 1-4. — С. 53–170.

2. Мальцев А.И. Алгоритмы и рекурсивные функции / А.И. Мальцев. — Москва : Наука, 1965. — 392 с.

3. Мартьянов В.И. Об инвариантных преобразованиях формул / В.И. Мартьянов // Математические заметки. — 1984. — Т. 36, вып. 4. — С. 571–581.

4. Van Hentenrick P. Constraint Satisfaction in Logic Programming / P. van Hentenrick. — London : The MIT Press, 1989. — 356 p.

5. Мартьянов В.И. Комбинаторные задачи высокой сложности и анализ плоских контурных изображений / В.И. Мартьянов, М.Д. Каташевцев // Известия Иркутского государственного университета. Серия: Математика. — 2013. — Т. 6, № 4. — С. 31–47.

6. Ершов Ю.Л. Математическая логика : учеб. пособие / Ю.Л. Ершов, Е.А. Палютин. — Москва : Наука, 1987. — 336 с.

7. Робинсон Дж. Машино-ориентированная логика, основанная на приципе резолюции / Дж. Робинсон // Кибернетический сборник (новая серия) / под ред. О.Б. Лупанова. — Москва, 1970. — Вып. 7. — С. 194–218.

8. Мартьянов В.И. О реализации схем доказательств в методе инвариантных преобразований / В.И. Мартьянов, А.А. Хармеев, Н.П. Яковлев // Кибернетика. — 1988. — № 3. — С. 78–83.

9. Александров С.Г. Применение системы АДТ для решения задач сетевого планирования / С.Г. Александров, В.И. Мартьянов // Интеллектуализация программных средств : сб. науч. тр. — Новосибирск, 1990. — С. 160–168.

10. Мартьянов В.И. Планирование информационных потоков в иерархической системе / В.И. Мартьянов, В.В. Сухорутченко, В.В. Окунцов // Прикладные системы. — Москва, 1992. — С. 46–58.

Информация об авторе

Мартьянов Владимир Иванович — доктор физико-математических наук, профессор, профконсультант, Байкальский государственный университет, г. Иркутск, Российская Федерация, e-mail: martvliv@mail.ru, SPIN-код: 5011-7030, Scopus Author ID: 16464234100.

Author

Vladimir I. Martyanov — D.Sc. in Physics and Mathematics, Professor, Professional Consultant, Baikal State University, Irkutsk, Russian Federation, e-mail: martvliv@mail.ru, SPIN-Code: 5011-7030, Scopus Author ID: 16464234100.

Для цитирования

Мартьянов В.И. NP-трудные задачи: автоматическое доказательство теорем и машины Тьюринга / В.И. Мартьянов. — DOI 10.17150/2411-6262.2021.12(4).11 // Baikal Research Journal. — 2021. — Т. 12, № 4.

For Citation

Martyanov V.I. NP-Difficult Tasks: Automatic Proof of Theorems and Turing's Machine. *Baikal Research Journal*, 2021, vol. 12, no. 4. (In Russian). DOI: 10.17150/2411-6262.2021.12(4).11.