О поиске вывода в системе натуральной дедукции логики предикатов

Ю. И. Вторушин

В статье описывается метод автоматического доказательства теорем, который используется в системах автоматизации дедукции Class и Int. Публикуемый алгоритм находит выводы в рамках многосортной системы натуральной дедукции, которая адекватно моделирует реальные человеческие рассуждения. Алгоритм полон для минимальной, интуиционистской и классической систем натуральной дедукции логики предикатов.

1. Синтаксис и семантика языков САД

Вначале дадим краткий обзор понятий и компонентов *систем автоматизации дедукции* (САД). Для этого, ниже будет определен язык, удобный для записи естественных рассуждений. Синтаксис рассмотренного ниже языка близок к языку систем автоматизации дедукции Class и Int ([1], [2], [3]).

Прежде всего, определим понятие *предикатной формулы*, которое является моделью повествовательного предложения, интуитивно воспринимаемого как истинное или ложное. Следующая грамматика $\mathfrak{G}_1$, записанная в нотации БНФ, определяет синтаксис формул для многосортного языка логики предикатов:

$$
\begin{array}{lcl}
\langle\text{формула}\rangle & \to & \langle\text{атомарная формула}\rangle \\
& | & \langle\text{кванторная формула}\rangle \\
& | & \langle\text{формула}\rangle \ \textbf{\&} \ \langle\text{формула}\rangle \\
& | & \langle\text{формула}\rangle \ \textbf{or} \ \langle\text{формула}\rangle \\
& | & \langle\text{формула}\rangle \ \textbf{implies} \ \langle\text{формула}\rangle \\
& | & \langle\text{формула}\rangle \ \textbf{iff} \ \langle\text{формула}\rangle
\end{array}
$$

	|	**not** ⟨формула⟩
	|	**true**
	|	**false**
	|	**contradiction**
	|	**thesis**
	|	**(** ⟨формула⟩ **)**
⟨кванторная формула⟩	→	**for** ⟨описанные переменные⟩ [**st** ⟨формула⟩] (**holds** ⟨формула⟩ | ⟨кванторная формула⟩)
	|	**ex** ⟨описанные переменные⟩ **st** ⟨формула⟩
⟨описанные переменные⟩	→	⟨список переменных⟩
	|	⟨явно описанные переменные⟩
	|	⟨явно описанные переменные⟩ **,** ⟨список переменных⟩
⟨явно опис. переменные⟩	→	⟨сегмент описаний⟩ {**,** ⟨сегмент описаний⟩}
⟨сегмент описаний⟩	→	⟨список переменных⟩ ⟨описание⟩
⟨описание⟩	→	(**being** | **be**) ⟨тип⟩

В качестве переменных, обозначающих произвольные формулы многосортного языка логики предикатов будем использовать прописные греческие буквы Φ, Ψ, Θ, ... Соответствие между традиционными обозначениями математической логики и обозначениями языка, заданного грамматикой $\mathfrak{G}_1$, дается следующей таблицей:

истина	$\top$	**true**
ложь	$\bot$	**false**
	$\bot$	**contradiction**
отрицание	$\neg\,\Phi$	**not** Φ
конъюнкция	$\Phi \,\&\, \Psi$	Φ **&** Ψ
дизъюнкция	$\Phi \vee \Psi$	Φ **or** Ψ
импликация	$\Phi \supset \Psi$	Φ **implies** Ψ
эквиваленция	$\Phi \equiv \Psi$	Φ **iff** Ψ
квантор всеобщности	$\forall X \Phi$	**for** X **holds** Φ
	$\forall X (\Phi \supset \Psi)$	**for** X **st** Φ **holds** Ψ
квантор существования	$\exists X \Phi$	**ex** X **st** Φ

Множество всех термов сигнатуры Ω с переменными из множества $\mathcal{X}$ будем обозначать через $\mathbf{Tm}_\Omega(\mathcal{X})$, а множество всех типов для Ω обозначим $\mathbf{Tp}_\Omega$. Будем также рассматривать множество *константных термов* (или просто *констант*), то есть термов без переменных, обозначаемое $\mathbf{Tm}_\Omega$.

Для любых сигнатуры Ω и множества переменных $\mathcal{X}$ для множества термов $\mathbf{Tm}_\Omega(\mathcal{X})$, а также для *функции типа* терма $\mathbf{type}(\tau)$ и *отношения подтипа* $\theta \preceq \theta'$, справедливо следующее утверждение: если $\tau_1, \ldots, \tau_n \in \mathbf{Tm}_\Omega(\mathcal{X})$, $f \in \Omega$, $\mathbf{arity}(f) = (\theta_1, \ldots, \theta_n)$, $\mathbf{range}(f) = \theta$ и $\mathbf{type}(\tau_i) \preceq \theta_i$, то $\tau = f(\tau_1, \ldots, \tau_n) \in \mathbf{Tm}_\Omega(\mathcal{X})$ и $\mathbf{type}(\tau) = \theta$.

В качестве *логических аксиом* системы натурального вывода логики предикатов принимаются символ истины $\top$ и формулы вида $t = t$ для всех термов t. Ниже перечисляются *правила вывода* для *классической* системы натуральной дедукции в виде удобном для их запоминания. С помощью символа «$*$», употребляемого при формулировании непрямых правил, отмечено *допущение* локального вывода. Обозначим классическую систему натуральной дедукции логики предикатов через $\mathfrak{D}_1$:

$$\&\text{в}\quad \frac{\Phi,\ \Psi}{\Phi \,\&\, \Psi} \qquad\qquad \&\text{и}\quad \frac{\Phi \,\&\, \Psi}{\Phi} \qquad \frac{\Phi \,\&\, \Psi}{\Psi}$$

$$\vee\text{в}\quad \frac{\Phi}{\Phi \vee \Psi} \qquad \frac{\Psi}{\Phi \vee \Psi} \qquad\qquad \vee\text{и}\quad \frac{\Phi \vee \Psi,\ \Phi \supset \Theta,\ \Psi \supset \Theta}{\Theta}$$

$$\supset\text{в}\quad \frac{\begin{array}{c}\Phi\ *\\ \vdots\\ \Psi\end{array}}{\Phi \supset \Psi} \qquad\qquad \supset\text{и}\quad \frac{\Phi \supset \Psi,\ \Phi}{\Psi}$$

$$\top\text{в}\quad \top \qquad\qquad \bot\text{и}\quad \frac{\begin{array}{c}\neg\,\Phi\ *\\ \vdots\\ \bot\end{array}}{\Phi}$$

$$\forall\text{в}\quad \frac{\begin{array}{c}x\ *\\ \vdots\\ \Phi(x)\end{array}}{\forall x \Phi(x)} \qquad\qquad \forall\text{и}\quad \frac{\forall x \Phi(x)}{\Phi(x/t)}$$

$$\exists\text{в}\quad \frac{\Phi(x/t)}{\exists x\Phi(x)} \qquad\qquad \exists\text{и}\quad \frac{\exists x\Phi(x)}{\Phi(x/y)}$$

$$=\text{в}\quad t=t \qquad\qquad =\text{и}\quad \frac{t{=}s,\ \Phi(t)}{\Phi(t//s)}$$

Заметим, что $\Phi \equiv \Psi$ есть сокращение для $(\Phi \supset \Psi)\&(\Psi \supset \Phi)$, таким образом, нам не нужны правила для связки $\equiv$. Также $\neg\Phi$ есть сокращение для $(\Phi \supset \bot)$. Кроме перечисленных правил требуется еще *правило реитерации*. Кроме того, важно отметить, что заменив выше указанное правило $\bot$и на одноименное изображенное ниже, получим систему *интуиционистской* логики предикатов $\mathfrak{D}_2$. Полностью удалив правило $\bot$и, получим систему *минимальной* логики $\mathfrak{D}_3$. Ниже формулируемые правила $\bot$и и $\neg\neg$и являются *производными* в классической системе и играют особую роль во взаимоотношениях между классической и интуиционистской логиками. Добавление к минимальной (или к интуиционистской) системе правила $\neg\neg$и делает ее равнообъемной классической.

$$\text{реитерация}\quad \frac{\Phi}{\Phi} \qquad\qquad \bot\text{и}\quad \frac{\bot}{\Phi} \qquad\qquad \neg\neg\text{и}\quad \frac{\neg\neg\Phi}{\Phi}$$

Для предикатных правил, необходимо сказать, что символом «$*$» в правиле $\forall$в отмечено введение новой *обобщаемой* переменной x, которая становится параметром рассуждения и ниже в тексте вывода может иметь свободные вхождения в формулы. Таким же свойством обладает правило вывода $\exists$и, которое фиксирует новую переменную (*локальную константу*) y. При этом, формула $\Phi\,(x/y)$ представляет собой результат *правильной подстановки* y вместо x в Φ. В правилах $\forall$и и $\exists$в также подразумеваются правильные подстановки. В правиле $=$и через $\Phi\,(t//s)$ обозначена формула, которая может быть получена из формулы Φ в результате замены некоторых вхождений (быть может всех) терма t на терм s.

Важно отметить, что, хотя многосортный генератор ищет вывод детализированный до уровня правил вывода (с последующей оптимизацией), тем не менее *алгоритм очевидности* верификатора САД, осуществляя проверку того, что одна формула является непосредственным следствием других формул, не должен детализироваться до уровня логических правил вывода. Точнее говоря, верификатор должен распознавать *отношение пропозиционального следования*. Это

может быть достигнуто с помощью алгоритма приведения формулы к тождественно-истинному виду нормальной формы (например, как в системе PC Mizar [7]).

Следующая грамматика $\mathfrak{G}_2$ определяет язык, удобный для моделирования системы натурального вывода многосортной логики предикатов:

⟨статья⟩	→	**environ** ⟨окружение⟩ **text** ⟨текст⟩
⟨окружение⟩	→	[⟨пункт окружения⟩]
	|	⟨пункт окружения⟩ **;** ⟨окружение⟩
⟨пункт окружения⟩	→	⟨описание переменных⟩
	|	⟨описание типа⟩
	|	⟨описание функции⟩
	|	⟨описание предиката⟩
	|	⟨суждение⟩
⟨описание переменных⟩	→	**reserve** ⟨сегмент описаний⟩ {**,** ⟨сегмент описаний⟩}
⟨сегмент описаний⟩	→	⟨список переменных⟩ **for** ⟨тип⟩
⟨описание типа⟩	→	**sort** ⟨тип⟩ [–> ⟨список типов⟩]
⟨описание функции⟩	→	**func** ⟨функция⟩ **(**⟨список типов⟩**)** –> ⟨тип⟩
⟨описание предиката⟩	→	**pred** ⟨предикат⟩ **[**⟨список типов⟩**]**
⟨text⟩	→	[⟨утверждение⟩]
	|	⟨утверждение⟩ **;** ⟨text⟩
⟨утверждение⟩	→	⟨распределенное утверждение⟩
	|	⟨компактное утверждение⟩
⟨распределен. утверждение⟩	→	[⟨метка⟩ **:**] **begin** ⟨рассуждение⟩ **end**
⟨компактное утверждение⟩	→	⟨суждение⟩ ⟨обоснование⟩
⟨суждение⟩	→	[⟨метка⟩ **:**] ⟨формула⟩
⟨обоснование⟩	→	⟨непосредственное обоснование⟩
	|	⟨доказательство⟩
⟨непосредств. обоснование⟩	→	[**by** ⟨список меток⟩]
⟨список меток⟩	→	⟨метка⟩ {**,** ⟨метка⟩}
⟨доказательство⟩	→	**proof** ⟨рассуждение⟩ **qed**
⟨рассуждение⟩	→	[⟨пункт рассуждения⟩]
	|	⟨пункт рассуждения⟩ **;** ⟨рассуждение⟩
⟨пункт рассуждения⟩	→	⟨утверждение⟩
	|	⟨допущение⟩
	|	⟨заключение⟩
	|	⟨генерализация переменных⟩
	|	⟨фиксация переменных⟩

	|	⟨итеративное равенство⟩
⟨допущение⟩	→	**assume** ⟨суждение⟩
⟨заключение⟩	→	**thus** ⟨утверждение⟩
⟨генерализация переменных⟩	→	**let** ⟨описанные переменные⟩ [**such** ⟨условия⟩]
⟨фиксация переменных⟩	→	**consider** ⟨описанные переменные⟩ [**such** ⟨условия⟩ ⟨непоср. обоснов.⟩]
⟨условия⟩	→	**that** ⟨суждение⟩ {**and** ⟨суждение⟩}
⟨итеративное равенство⟩	→	⟨терм⟩ = ⟨терм⟩ ⟨непоср. обоснов.⟩ **.**= ⟨терм⟩ ⟨непоср. обоснов.⟩ {**.**= ⟨терм⟩ ⟨непоср. обоснов.⟩}

Согласно сформулированному определению, рассуждения (выводы) языка дедуктирования $\mathbf{L}(\mathfrak{G}_2)$ имеют отчетливую субординатную структуру, что имеет принципиальное значение для систем натуральной дедукции. Действительно, всякий вывод является последовательностью допущений, заключений и утверждений. В свою очередь, эти утверждения и заключения сами могут включать рассуждения, которые, таким образом, являются *подвыводами* исходного вывода. Семантика языка подразумевает сопоставление каждому рассуждению (выводу), которое представляет собой распределенное утверждение или доказательство, определенной формулы, которая называется *тезисом* этого рассуждения. При этом, в рамках таких представлений, подразумевается, что непрямых правил вывода имеется бесконечное количество, среди которых непрямые правила $\supset$в, $\bot$и и $\forall$в составляют только три отдельных частных случая.

Пусть Φ_1, Φ_2, …, Φ_n, Ψ — список (по порядку) всех допущений и заключений вывода Π, где Ψ — последнее заключение[1]. *Тезисом* (*прямым тезисом*) рассуждения Π называется следующая формула

$$\forall\bar{x}_1(\Phi_1\,\lambda_1\,\forall\bar{x}_2(\Phi_2\,\lambda_2\,(\ldots\forall\bar{x}_n(\Phi_n\,\lambda_n\,\forall\bar{x}_{n+1}\Psi))\ldots),$$

[1]Необходимо, чтобы список Δ всех допущений и заключений рассуждения Π всегда завершался заключением. Поэтому, в том случае, когда Δ является пустым или последней формулой является допущение, то удобно (по умолчанию) считать, что последним заключением в этом списке является символ истины $\top$, то есть в этом случае в качестве списка всех допущений и заключений для Π подразумевается $\Delta\top$.

где $\bar{x}_i$ — список всех переменных из пунктов генерализаций, находящихся между пунктами Φ_{i-1} и Φ_i; а связка λ_i вычисляется следующим образом

$$\lambda_i = \begin{cases} \&, & \text{если } \Phi_i \text{ — заключение,} \\ \supset, & \text{если } \Phi_i \text{ — допущение.} \end{cases}$$

Для доказательств в рамках классической логики, в случае, когда последним заключением Ψ рассуждения Π является противоречие $\bot$, а также Φ_n имеет вид $\neg\,\Theta$ и λ_n есть $\supset$, помимо прямого тезиза, рассматривают также *косвенный тезис*

$$\forall\bar{x}_1(\Phi_1\,\lambda_1\,\forall\bar{x}_2(\Phi_2\,\lambda_2\,(\ldots\forall\bar{x}_{n-1}(\Phi_{n-1}\,\lambda_{n-1}\,\forall\bar{x}_n\Theta))\ldots).$$

Алгоритм семантической корректности верификатора последовательно проверяет, во-первых, правильность непосредственных умозаключений, которые соответствуют применению прямых правил вывода; во-вторых, правильность структуры доказательств, что соответствует применению непрямых правил вывода. В последнем случае проверяется соответствие тезиса доказательства доказываемой формуле.

Наиболее важная роль типов переменных в многосортном языке САД заключается в следующем.

1) Когда переменная упомянута в статье, должен быть описан ее тип.
2) У каждого терма τ статьи верификатор вычисляет $\mathbf{type}(\tau)$ — уникальный *тип* терма τ.
3) Один тип θ_1 может быть *подтипом* другого θ_2 (символически, $\theta_1 \preceq \theta_2$). В этом случае также говорят, что тип θ_1 *расширяется* до θ_2, а также, что тип θ_2 есть *надтип* типа θ_1. Отношение $\preceq$ является частичным порядком и связывает типы данных в иерархию наследования, которая позволяет применять правила редукции термов типа θ к термам-данным с более узким типом $\theta' \preceq \theta$. Говорят, что терм τ *имеет тип* θ, если тип $\mathbf{type}(\tau)$ расширяется к θ.

4) В процессе *автоматической редукции* терма τ, когда в результате выполнения программы автоматического доказательства теорем вычисляется *нормальная форма* терма $\tau = \tau_0 \Rightarrow \tau_1 \Rightarrow \ldots$, тип терма не меняется, оставаясь неизменным (то есть является инвариантом). Это означает, что если $\tau \Rightarrow^* \tau'$, то $\mathbf{type}(\tau) = \mathbf{type}(\tau')$.

В следующем примере показывается, что высказывание

«Каждый ученик второго класса прочитал хотя бы одну книгу»

является следствием высказывания

«Существует книга, которую прочитал каждый ученик второго класса».

```
environ
  sort Book;
  sort Pupil;
  pred HadRead[Pupil, Book];
  A: ex W being Book st for Y being Pupil holds HadRead[Y,W];
text

  for X being Pupil ex V being Book st HadRead[X,V]
proof let X being Pupil;
  consider W being Book such that
B:  for Y being Pupil holds HadRead[Y,W] by A;
C:  HadRead[X,W] by B;
  thus ex V being Book st HadRead[X,V] by C;
qed;
```

Текст этого примера можно сделать короче, если использовать резервирование переменных.

```
environ
  sort Book;
  sort Pupil;
  pred HadRead[Pupil, Book];
  reserve X, Y for Pupil;
  reserve V, W for Book;
  A: ex W st for Y holds HadRead[Y,W];
```

```
text

  for X ex V st HadRead[X,V]
proof let X;
  consider W such that B: for Y holds HadRead[Y,W] by A;
  C: HadRead[X,W] by B;
  thus ex V st HadRead[X,V] by C;
qed;
```

2. Иерархия типов и редукция термов

Необходимым условием эффективной работы САД является *разрешимость* системы понятий, постулируемых пользователем с помощью аксиоматических спецификаций. Система пользовательских понятий, которые регистрируются в рамках таких спецификаций, включает *типы переменных*, *функциональные* и *предикатные символы* языка формализуемой теории. Такая система понятий составляет сигнатуру Ω многосортного языка логики предикатов моделируемой предметной области. Кроме разрешимости, важными вопросами являются *корректность* и *полнота* такой системы понятий. Ниже будут рассмотрены достаточные условия разрешимости системы типов и разрешимости эквациональной теории для пользовательских спецификаций.

Известно, что когда в САД типы имеют аргументы[2], то отношение подтипа в общем случае является неразрешимым, то есть не существует алгоритма, эффективно выясняющего является ли один тип подтипом другого. Но наложение некоторых ограничений на систему типов позволяет выделить удобный для нас подкласс спецификаций, для которых отношение подтипа разрешимо.

Множество типов $\mathbf{Tp}_\Omega$ языка сигнатуры Ω вместе с отношением частичного порядка $\preceq$, порожденное системой спецификаций $\mathfrak{A}$, считается *нетеровой верхней полурешеткой*, коль скоро выполняются следующие два свойства.

[2]Например, развитая система зависимых полиморфных типов существует в системе PC Mizar [4].

1) Множество типов $\mathbf{Tp}_\Omega$ есть *верхняя полурешетка*. Это означает, что каждая пара типов $\{\theta_1, \theta_2\}$ имеет наименьшую верхнюю грань $\theta_1 \sqcup \theta_2$. То есть, во-первых, $\theta_1 \preceq \theta_1 \sqcup \theta_2$ и $\theta_2 \preceq \theta_1 \sqcup \theta_2$; во-вторых, если $\theta_1 \preceq \theta$ и $\theta_2 \preceq \theta$, то $\theta_1 \sqcup \theta_2 \preceq \theta$ для любого типа θ.
2) Отношение $\preceq$ является *полным порядком* (*нетеровость*). Это свойство означает, что каждое непустое множество типов $\mathcal{T} \subseteq \mathbf{Tp}_\Omega$ имеет максимальный элемент. Это означает, также, что не существует бесконечной последовательности типов $\theta_1 \prec \theta_2 \prec \theta_3 \prec \ldots$

В каждой нетеровой верхней полурешетке любой идеал $\mathcal{I}$ имеет наименьшую верхнюю грань: $\mathbf{sup}\,\mathcal{I} \in \mathcal{I}$. Таким образом, если $\mathbf{Tp}_\Omega$ есть нетерова верхняя полурешетка, то любой идеал $\mathcal{I} \subseteq \mathbf{Tp}_\Omega$ имеет точную верхнюю грань. В частности, множество $\downarrow \theta = \{\theta' \in \mathbf{Tp}_\Omega : \theta' \preceq \theta\}$ является идеалом, имеющего своей точной верхней гранью тип θ.

Наибольший (над)тип в $\mathbf{Tp}_\Omega$ (то есть верхняя грань полурешетки) обозначается $\top$. Примерами таких наибольших типов могут служить: `Nat` — тип «натуральное число» в арифметике Пеано, `Set` — тип «множество» в теории множеств Цермело–Френкеля и др.

В каждой нетеровой верхней полурешетке множество $\uparrow \theta = \{\theta' : \theta \preceq \theta'\}$ конечно. Таким образом, в нетеровой верхней полурешетке типов множество $\uparrow \theta = \{\theta' \in \mathbf{Tp}_\Omega : \theta \preceq \theta'\}$ конечно и может быть эффективно вычислено.

Отсюда вытекает, что когда система типов является нетеровой верхней полурешеткой, отношение подтипа разрешимо; для выяснения, является ли тип θ_1 подтипом типа θ_2, достаточно найти все надтипы типа θ_1. Ясно, что этот же алгоритм может быть применен для нахождения всех типов заданного терма.

К сожалению, в общем случае, когда полиморфные параметрические типы САД образуют древовидную иерархию типов (как, например, в [4]), проверка по данной системе спецификаций, является ли соответствующая система типов нетеровой верхней полурешеткой, сама по себе неразрешимая проблема. Но имеется ряд достаточных условий, которые обеспечивают для системы типов это свойство.

Что касается равенства термов, то известно, что даже в односортном случае эквациональная теория, одновременно с отношением редукции, в общем случае являются неразрешимыми, то есть не существует алгоритма, эффективно выясняющего эквивалентность, или редуцируемость одного к другому, двух предъявленных ему термов. Но наложение некоторых ограничений на отношение редукции позволяет выделить удобный для нас подкласс спецификаций, для которых эквациональная теория разрешима.

Имея в виду контекст автоматического доказательства теорем, опишем эквациональную семантику языка САД точнее. Аксиоматическая спецификация $\mathfrak{A}$ определяет на множестве константных термов $\mathbf{Tm}_\Omega$ языка сигнатуры Ω *отношение редукции* $\Rightarrow$, определяющееся следующим образом: $\tau_1 \Rightarrow \tau_2$ в том случае, если в $\mathfrak{A}$ существует постулированное в виде аксиомы, или полученное пользователем как теорема, тождество $\lambda = \rho$ (λ — левая часть, а ρ — правая часть этого тождества, $\mathbf{vars}(\rho) \subseteq \mathbf{vars}(\lambda)$), и существует функция конкретизации $\sigma : \mathbf{vars}(\lambda) \to \mathbf{Tm}_\Omega$ такая, что τ_1 содержит $\lambda\sigma$ как подтерм и τ_2 получается из τ_1 путем замены этого подтерма на $\rho\sigma$ (при этом не должен измениться тип: $\mathbf{type}(\lambda\sigma) = \mathbf{type}(\rho\sigma)$). Если в качестве $\lambda\sigma$ выступает сам терм τ_1, то говорят, что данная аксиома $\lambda = \rho$ применима ко *всему* терму.

Будем обозначать $\Rightarrow^*$ — транзитивно-рефлексивное замыкание, а $\Leftrightarrow^*$ — транзитивно-рефлексивно-симметричное замыкание отношения $\Rightarrow$. Множество всех констант $\mathbf{Tm}_\Omega$ является носителем некоторой Ω-алгебры, называемой алгеброй Ω-термов. В качестве эквациональной семантики аксиоматической спецификации $\mathfrak{A}$ понимают факторалгебру алгебры Ω-термов по эквивалентности $\Leftrightarrow^*$. Равенство $\tau_1 = \tau_2$ выводимо (или истинно) тогда и только тогда, когда $\tau_1 \Leftrightarrow^* \tau_2$. Множество всех выводимых равенств вида $\tau_1 = \tau_2$ называется *эквациональной теорией* аксиоматической спецификации $\mathfrak{A}$.

Пусть имеется система аксиом $\mathfrak{A}$, которая порождает на множестве константных термов $\mathbf{Tm}_\Omega$ отношение редукции $\Rightarrow$. Константа τ называется *редуцируемой*, если существует константа τ' такая, что $\tau \Rightarrow \tau'$, и *нередуцируемой* в противном случае. Константа τ' называется *нормальной формой* константы τ, если $\tau \Rightarrow^* \tau'$ и τ' нередуцируемая.

Определения отношения редукции и редуцируемости естественным образом распространяются на термы с переменными. Например, терм `len(cons(Y, rev(L)))` редуцируем согласно аксиоме `len(cons(X, W))` = `s(len(W))` и результатом редукции будет `s(len(rev(L)))`.

Напомним свойства конфлюентности и нетеровости, применив их к отношению $\Rightarrow$. Отношение редукции $\Rightarrow$, порожденное системой аксиом $\mathfrak{A}$, будем называть:

1) *нетеровым*, если для любого терма $\tau \in \mathbf{Tm}_\Omega$ не существует бесконечной последовательности редукций $\tau \Rightarrow \tau_1 \Rightarrow \tau_2 \Rightarrow \ldots$
2) *конфлюентым*, если при условии, что имеет место $\tau \Rightarrow^* \tau_1$ и $\tau \Rightarrow^* \tau_2$ обязательно найдется τ_3 такой, что $\tau_1 \Rightarrow^* \tau_3$ и $\tau_2 \Rightarrow^* \tau_3$. Здесь τ, τ_1, τ_2, $\tau_3 \in \mathbf{Tm}_\Omega$.

Верно следующее утверждение: если отношение редукции является нетеровым и конфлюентным, то для любой константы τ существует единственная нормальная форма, которая обозначается $\|\tau\|$.

Связь между эквивалентностью и так называемым свойством Черча–Россера указывается в следующей теореме. Если отношение редукции $\Rightarrow$ является конфлюентным, то $\tau_1 \Leftrightarrow^* \tau_2$ тогда и только тогда, когда существует терм τ_3 такой, что $\tau_1 \Rightarrow^* \tau_3$ и $\tau_2 \Rightarrow^* \tau_3$. Здесь термы τ_1, τ_2, $\tau_3 \in \mathbf{Tm}_\Omega$.

Отсюда получаем важное следствие. Пусть отношение редукции $\Rightarrow$ является конфлюентным и нетеровым. Тогда $\tau_1 \Leftrightarrow^* \tau_2$ тогда и только тогда, когда их нормальные формы совпадают: $\|\tau_1\| = \|\tau_2\|$.

Таким образом, когда отношение редукции является конфлюентным и нетеровым, оно является разрешимым вместе с эквациональной теорией; для выяснения эквивалентности двух термов необходимо лишь выяснить совпадение их нормальных форм. Так как отношение редукции является нетеровым, а система аксиом конечна, процесс нахождения нормальной формы эффективен.

К сожалению, проверка по данной системе аксиом, является ли соответствующее отношение редукции конфлюентным и нетеровым, сама по себе неразрешимая проблема. Но имеется ряд достаточных условий для выполнения таких свойств.

Полученное следствие позволяет сделать еще один важный вывод, что в случае конфлюентного и нетерового отношения редукции множество нормальных форм находится во взаимно-однозначном соответствии со множеством классов эквивалентности. Таким образом, согласно определению семантики, в качестве объекта абстрактного типа можно рассматривать множество термов, нормальные формы которых совпадают.

Например, рассмотрим следующую аксиоматическую спецификацию $\mathfrak{A}_1$ типа «список натуральных чисел», который обозначим `List`. Среди определяемых ниже функций, функции `nil()` и `cons(X,L)` являются *базовыми конструкторами* этого типа.

```
environ

  sort Nat;
  func o() -> Nat;
  func s(Nat) -> Nat;

  sort List;
  func nil() -> List;
  func cons(Nat, List) -> List;
  func conc(List, List) -> List;
  reserve X for Nat, W, U for List;
  1: for U holds conc(nil(), U) = U;
  2: for X,U,W holds conc(cons(X, U), W) = cons(X, conc(U, W));

  func conr(List, Nat) -> List;
  3: for X,W holds conr(W, X) = conc(W, cons(X, nil()));

  func rev(List) -> List;
  4: rev(nil()) = nil();
  5: for X,W holds rev(cons(X, W)) = conr(rev(W), X);
```

Например, терм `rev(cons(o, cons(s(o), nil)))`, в соответствии с этим определением функций, имеет тип `List`. В результате редукции получаем последовательность термов типа `List`:

```
rev(cons(o, cons(s(o), nil)))
     ⇒ conr(rev(cons(s(o), nil)), o)
     ⇒ conc(rev(cons(s(o), nil)), cons(o, nil))
     ⇒ conc(conr(rev(nil), s(o)), cons(o, nil))
```

```
    ⇒ conc(conc(rev(nil), cons(s(o), nil)), cons(o, nil))
    ⇒ conc(conc(nil, cons(s(o), nil)), cons(o, nil))
    ⇒ conc(cons(s(o), nil), cons(o, nil))
    ⇒ cons(s(o), conc(nil, cons(o, nil)))
    ⇒ cons(s(o), cons(o, nil)).
```

В рамках языка САД последнее рассуждение может быть записано в виде следующего итеративного равенства:

```
text

  rev(cons(o(), cons(s(o()), nil()))) =
                                cons(s(o()), cons(o(), nil()))
proof
  6: rev(cons(s(o()), nil())) = conr(rev(nil()), s(o())) by 5;
  7: conr(rev(nil()), s(o())) =
                     conc(rev(nil()), cons(s(o()), nil())) by 3;
  8: conc(nil(), cons(s(o()), nil())) = cons(s(o()), nil()) by 1;
  9: conc(nil(), cons(o(), nil())) = cons(o(), nil()) by 1;
  thus rev(cons(o(), cons(s(o()), nil())))
    = conr(rev(cons(s(o()), nil())), o()) by 5
   .= conc(rev(cons(s(o()), nil())), cons(o(), nil())) by 3
   .= conc(conr(rev(nil()), s(o())), cons(o(), nil())) by 6
   .= conc(conc(rev(nil()), cons(s(o()), nil())),
                                          cons(o(), nil())) by 7
   .= conc(conc(nil(), cons(s(o()), nil())),
                                          cons(o(), nil())) by 4
   .= conc(cons(s(o()), nil()), cons(o(), nil())) by 8
   .= cons(s(o()), conc(nil(), cons(o(), nil()))) by 2
   .= cons(s(o()), cons(o(), nil())) by 9;
qed;
```

В дальнейшем пользователем могут быть сконструированы новые типы, при этом, некоторые из функций могут быть переопределены. Например, в качестве дочерних подтипов для типа `List` можно ввести типы: «пустой список» `EmpList`, «непустой список» `NonEmpList`, «упорядоченный список» `OrdList` и др.

3. Автоматическое доказательство теорем

Многосортный язык системы натуральной дедукции логики предикатов представляет собой естественный фундамент для современной технологии хранения и обработки информации с помощью компьютера. Удобный с практической точки зрения язык прикладной САД, предназначенный для хранения и обработки семантически проверенной логико-математической базы знаний, должен обладать следующими свойствами:

1) высокий уровень синтаксического развития языка, дающий возможность представления разнообразных форм *атомарных выражений* (то есть разные префиксные, инфиксные и постфиксные формы записи термов и предикатов, функциональные скобки и др.);
2) удобные, близкие к естественным, формы выражения *типов* (сортов) переменных, которые могут образовывать древовидные иерархии подчиненных типов;
3) возможность делать *определения* новых понятий, которые вводят в язык пользовательские типы, функциональные и предикатные символы;
4) возможность формулировать и доказывать *схемы* теорем, содержащие предикатные и функциональные переменные, вместо которых могут подставляться конкретные формулы[3].
5) *оптимизация* и *автоматическое доказательство* является важной частью САД, позволяющее пользователю оптимизировать и автоматически *генерировать* фрагменты текстов своих статей.

В этой работе мы рассматриваем новый алгоритм поиска доказательства формулы, который был разработан автором. Исторически, первым пытался найти такую процедуру Лейбниц (1646–1716), в дальнейшем возобновили попытки Пеано (примерно на грани XX века) и школа Гильберта в 1920-х годах. Это продолжалось до тех пор,

[3]В этом случае в системах дедукции добавлются *правила подстановки*. В этих системах вывода такие доказуемые схемы теорем являются обоснованиями для *производных* правил вывода.

пока Черч и Тьюринг независимо не доказали, что не существует никакой общей разрешающей процедуры, никакого алгоритма, устанавливающего выводимость (или общезначимость[4]) формул в логике предикатов.

Тем не менее существуют алгоритмы поиска доказательства, а также установления выводимости, которые могут подтвердить выводимость (или общезначимость) формулы, если такая выводимость действительно имеет место (*полуразрешимость исчисления предикатов*). Для невыводимых (или необщезначимых) формул эти алгоритмы, вообще говоря, не заканчивают свою работу[5]. Принимая во внимание результат Черча и Тьюринга о неразрешимости исчисления предикатов, это лучшее, что мы можем ожидать от алгоритма поиска доказательства.

Заметим, что многие современные прикладные САД не содержат автоматического доказательства теорем, хотя включают эффективные алгоритмы *установления выводимости* в рамках алгоритма очевидности САД. Пока только некоторые из существующих систем автоматизации дедукции способны автоматически получать доказательства, которые пользователь может вставлять в свой текст формализованной статьи.

Ниже в этой работе рассматривается оригинальный алгоритм автоматического доказательства теорем, который используется в САД Class и Int ([1], [2], [3]). Подобно алгоритму, используемому в САД AProS ([5], [6]), этот алгоритм находит только *канонические нормальные выводы*[6]. Соответствующий вариант алгоритма полон для минимальной, интуиционистской и классической систем натуральной дедукции логики предикатов.

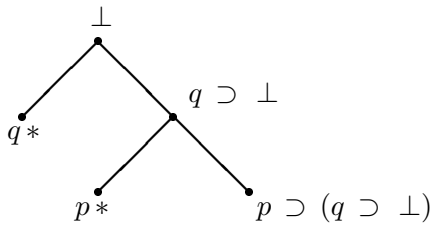
---

[4]Заметим, что в рамках классической логики метаутверждение $\Phi \vdash \Psi$ равносильно $\vdash (\neg\Phi) \vee \Psi$, а также равносильно $\Phi\&\neg\Psi \vdash \bot$.

[5]С практической точки зрения это означает, что прикладная программа аварийно завершает процесс поиска вывода после переполнения выделяемой области памяти.

[6]В САД AProS используется система натуральной дедукции с прямым правилом $\forall$в и непрямым правилом $\exists$и. В то время как, в этой статье (и в САД Class и Int) система натуральной дедукции содержит непрямое правило $\forall$в и прямое правило $\exists$и.

Чтобы составить представление об алгоритме генератора, приведем в качестве примера описание процесса решения простой дедуктивной задачи $p \,\&\, q \vdash \neg(p \supset \neg\, q)$. Точнее говоря, задачи $p \,\&\, q \vdash (p \supset (q \supset \bot)) \supset \bot$. Алгоритм решения дедуктивной задачи сводит ее к решению новых дедуктивных подзадач, при этом реализуется поиск вывода в глубину, сопровождаемый бэктрекингом. Чтобы ограничить дерево поиска, введен пользовательский параметр, устанавливающий максимальную глубину дерева поиска. На первом шаге решения нашей задачи получаем подзадачу $p \,\&\, q,\ p \supset (q \supset \bot) \vdash \bot$. Затем, для целевой формулы $\bot$, строится дерево поиска, отвечающее позитивному вхождению этой цели $\bot$ в посылку $p \supset (q \supset \bot)$:



Здесь символом $*$ отмечены новые вспомогательные подцели p и q, возникающие из негативных частей этой посылки, к которым вновь применяется описанный выше процесс. На следующем шаге цели p и q обнаруживаются в позитивных частях посылки $p \,\&\, q$, которая их непосредственно обосновывает. Наконец, в результате успешного решения этой дедуктивной задачи, генератор формирует из полученного в оперативной памяти компьютера дерева поиска следующий текст:

```
environ
  pred p[];
  pred q[];
  1: p[] & q[];
text

  not (p[] implies not q[])
proof
  assume 2: p[] implies not q[];
```

```
  3: p[] by 1;
  4: not q[] by 2,3;
  5: q[] by 1;
  thus false by 4,5;
qed;
```

В случае логики предикатов, разработанный алгоритм использует синтаксические метапеременные, а также вспомогательные сколемовские функциональные символы. При этом, локальные метапеременные и сколемовские функции ставятся в позитивную часть посылок текущей дедуктивной задачи, а глобальные метапеременные возникают в консеквенте этой задачи, отвечая обратному применению правила введения квантора существования $\exists$в. Метапеременным и сколемовским функциям приписывается их уровень в дереве поиска, который является ограничивающим фактором при поиске благоприятных подстановок. Каждая благоприятная подстановка $[X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n]$ задает в качестве значения метапеременным термы, построенные из предметных переменных и функциональных символов, с которыми работает генератор в текущей части дерева поиска. Наличие хотя бы одной благоприятной подстановки у исходной дедуктивной задачи означает успешное решение этой задачи поиска вывода. Искомый вывод получается из построенного дерева поиска в результате замещения всех метапеременных соответствующими термами исходного языка.

Следующий логический закон $\exists x(drinker[x] \supset \forall y\, drinker[y])$, называемый «парадоксом пьяницы», получен автоматически:

```
environ
  sort Human;
  pred drinker[Human];
  reserve x, y, t, tt, ttt for Human;
text

  ex x st (drinker[x] implies for y holds drinker[y])
proof
  assume 1: not ex x st drinker[x] implies for y holds drinker[y];
  consider t;
  2: drinker[t] implies for y holds drinker[y]
    proof
      assume 2: drinker[t];
```

```
      let tt;
      3: drinker[tt]
      proof
        assume 3: not drinker[tt];
        4: drinker[tt] implies for y holds drinker[y]
        proof
          assume 4: drinker[tt];
          let ttt;
          5: drinker[ttt]
          proof
            assume 5: not drinker[ttt];
            thus 6: false by 3,4;
          qed;
          thus drinker[ttt] by 5;
        qed;
        5: ex x st drinker[x] implies for y holds drinker[y] by 4;
        thus 6: false by 1,5;
      qed;
      thus drinker[tt] by 3;
    qed;
  3: ex x st drinker[x] implies for y holds drinker[y] by 2;
  thus 4: false by 1,3;
qed;
```

Заметим, что многосортный генератор ищет вывод детализированный до уровня правил вывода. Для эффективной работы с равенством, генератор использует посылки, имеющие вид тождеств, как правила редукции термов. Чтобы составить представление об эквациональных результатах работы генератора, ниже приведено в качестве примера доказательство теоремы Пифагора «если $\vec{a} \cdot \vec{b} = 0$, то $(\vec{a} + \vec{b})^2 = \vec{a}^2 + \vec{b}^2$», полученное автоматически:

```
environ

  sort Real;
  func s(Real, Real) -> Real;
  func o() -> Real;
  reserve x for Real;
  1: for x holds s(x,o()) = x;
  2: for x holds s(o(),x) = x;

  sort Vector;
  func f(Vector, Vector) -> Vector;
```

```
  func h(Vector, Vector) -> Real;
  reserve u,v,w for Vector;
  3: for u,v,w holds h(f(u,v),w) = s(h(u,w),h(v,w));
  4: for u,v,w holds h(u,f(v,w)) = s(h(u,v),h(u,w));

text

  for a, b being Vector st h(a,b) = o() & h(b,a) = o()
    holds h(f(a,b),f(a,b)) = s(h(a,a),h(b,b))
proof
  let a be Vector,b be Vector;
  assume 5: h(a,b) = o() & h(b,a) = o();
  6: h(f(a,b),f(a,b)) = s(h(a,f(a,b)),h(b,f(a,b))) by 3;
  7: h(a,f(a,b)) = s(h(a,a),h(a,b)) by 4;
  8: h(a,b) = o() by 5;
  9: s(h(a,a),o()) = h(a,a) by 1;
  10: h(b,f(a,b)) = s(h(b,a),h(b,b)) by 4;
  11: h(b,a) = o() by 5;
  12: s(o(),h(b,b)) = h(b,b) by 2;
  thus h(f(a,b),f(a,b)) = s(h(a,f(a,b)),h(b,f(a,b))) by 6
    .= s(s(h(a,a),h(a,b)),h(b,f(a,b))) by 7
    .= s(s(h(a,a),o()),h(b,f(a,b))) by 8
    .= s(h(a,a),h(b,f(a,b))) by 9
    .= s(h(a,a),s(h(b,a),h(b,b))) by 10
    .= s(h(a,a),s(o(),h(b,b))) by 11
    .= s(h(a,a),h(b,b)) by 12;
qed;
```

4. Алгоритм поиска вывода

Следующая процедура **решить**(Γ, Φ) решает задачу вида $\Gamma \vdash \Phi$ для классической многосортной системы натуральной дедукции $\mathfrak{D}_1$. Для этого создается последовательность Σ генерализаций, допущений и заключений. Эта последовательность просматривается по порядку, допущения вставляются в верхний лес, а для заключений строится вывод в виде нижнего дерева, растущего снизу вверх к допущениям верхнего леса. Здесь Γ — лес формул, возможно, содержащих локальные и глобальные метапеременные; Φ — формула, не содержащая локальные, и возможно, содержащая глобальные метапеременные. Возвращается набор подстановок.

процедура **решить**(Γ, Φ)
    $S := \varnothing$
    для каждого пункта Π последовательности Σ **декомпозировать**(Φ)
        выбор
            если Π имеет вид (`assume` Ψ)
                замкнуть(Γ, Ψ)
            если Π имеет вид (`thus` Ψ)
                $S = S \cup$ **вывести**(Γ, Ψ) $\cup$ **снятие-или2**(Γ, Ψ)
        конец выбор
    конец для
возврат S

Процедура **замкнуть**(Δ, Φ) добавляет в верхний лес Δ формулу Φ. При вставке импликации рекурсивно вставляются также их консеквенты. При вставке конъюнкции рекурсивно вставляются также оба ее члена. При вставке квантора общности рекурсивно вставляется также подкванторное выражение с заменой связанных переменных на локальные. При вставке квантора существования рекурсивно вставляется также подкванторное выражение с заменой связанных переменных на сколемовские функции.

процедура **замкнуть**(Δ, Φ)
вставить в Δ формулу Φ
выбор
    если Φ имеет вид (Ψ `imp` Θ)
        вставить в Δ формулу Θ как потомка Φ
    если Φ имеет вид (Ψ `&` Θ)
        вставить в Δ формулы Ψ и Θ как потомков Φ
    если Φ имеет вид (`for` $x_1, \ldots, x_m$ `holds` Ψ)
        вставить в Δ формулу Ψ $\left[\begin{array}{l} x_1 \leftarrow \text{новая лок. перем. типа } \mathbf{type}(x_1), \\ x_2 \leftarrow \text{новая лок. перем. типа } \mathbf{type}(x_2), \\ \ldots \\ x_m \leftarrow \text{новая лок. перем. типа } \mathbf{type}(x_m) \end{array}\right]$
            как потомка Φ
    если Φ имеет вид (`ex` $x_1, \ldots, x_m$ `st` Ψ)
        пусть $n = |\mathbf{lvar}(\Phi)|$
        ввести новые функциональные символы $f_1, \ldots, f_m$ местности n
            типов $\mathbf{type}(x_1), \ldots, \mathbf{type}(x_m)$ соответственно
        вставить в Δ формулу Ψ $\left[\begin{array}{l} x_1 \leftarrow f_1(t_1, \ldots, t_n), \\ x_2 \leftarrow f_2(t_1, \ldots, t_n), \\ \ldots \\ x_m \leftarrow f_m(t_1, \ldots, t_n) \end{array}\right]$ как потомка Φ,
            где $t_i \in \mathbf{lvar}(\Phi)$ $(i = 1, \ldots, n)$
конец выбор

Процедура-генератор **декомпозировать** возвращает структуру вывода для формулы Φ в виде последовательности генерализаций, допущений и заключений. Для квантора общности вводится новая предметная переменная, возвращается генерализация, затем продолжает декомпозироваться подкванторная формула с заменой связанной переменной на вновь введенную. Для импликации возвращается допущение антецедента, и продолжает декомпозироваться консеквент. Для конъюнкции возвращается заключение ее левого члена, и продолжает декомпозироваться ее правый член. Остальные формулы возвращаются в виде заключения.

```
процедура декомпозировать(Φ)
цикл
    выбор
        если Φ имеет вид (for x holds Ψ)
            ввести новую переменную t типа type(x)
            выдать (let t)
            Φ := Ψ[x ← t]
        если Φ имеет вид (Ψ imp Θ)
            выдать (assume Ψ)
            Φ := Ψ
        если Φ имеет вид (Ψ & Θ)
            выдать (thus Ψ)
            Φ := Θ
        иначе
            выдать (thus Φ)
            выход из цикл
    конец выбор
конец цикл
```

Процедура **вывести** строит нижнее дерево для формулы Φ, используя верхний лес Δ. Если Φ является аксиомой, возвращается набор подстановок с пустой подстановкой. Иначе инициализируем $S = \varnothing$ и проверяем ряд вариантов:

1) если Φ имеет вид $t_1 = t_2$ и σ является наиболее общим унификатором t_1 и t_2, то добавим σ в S.
2) если Φ — конъюнкция, то находим подстановки для ее членов, и их композицию добавим в S.
3) если Φ — дизъюнкция, то находим подстановки для ее членов, и добавим обе подстановки в S.

4) если Φ начинается с квантора существования, то находим подстановку для подкванторной формулы с заменой связанной переменной на новую глобальную переменную, и добавим ее в S.
5) если Φ начинается с квантора всеобщности или импликация, решаем задачу $\Gamma \vdash \Phi$ и найденные подстановки добавим в S.

Кроме того, если Φ — атом или дизъюнкция или противоречие, то пробуем обосновать Φ через верний лес. Для этого находим вершины верхнего леса такие, что их формулы унифицируются с Φ, и пользуемся процедурой **раскрутить**, чтобы построить поддерево нижнего дерева на основе обнаруженной ветви верхнего дерева и получить подстановки. Кроме того, атомы, дизъюнкции и противоречия пробуем обосновать по правилу снятия дизъюнкции. Возвращается набор подстановок.

процедура **вывести**(Δ, Φ)
пусть $S := \varnothing$ :: множество подстановок
если если Φ имеет вид (`true`) или ($t = t$)
    $S := S \cup \{\varepsilon\}$ :: добавили в множество подстановок пустую подстановку
иначе
    пусть $\sigma := \varepsilon$
    если Φ имеет вид ($t_1 = t_2$) и **унифицировать**(t_1, t_2, σ)
        $S := S \cup \{\sigma\}$
    конец если

    выбор
        если Φ имеет вид (Ψ `&` Θ)
            $S := S \cup$ **комбинировать**(**вывести**(Δ, Ψ), **вывести**(Δ, Θ))
        если Φ имеет вид (Ψ `or` Θ)
            $S := S \cup \{$**вывести**(Δ, Ψ), **вывести**(Δ, Θ)$\}$
        если Φ имеет вид (`ex` x `st` Ψ)
            $S := S \cup \{$**вывести**(Δ, $\Psi[x \leftarrow$ новая глоб. перем. типа **type**(x)$]$)$\}$
        если Φ имеет вид (`for` x `holds` Ψ) или (Ψ `imp` Θ)
            $S := S \cup \{$**решить**(Δ, Φ)$\}$
    конец выбор

    если Φ — атом или (имеет вид ($t_1 = t_2$) или (Ψ `or` Θ) или (`false`))
        :: эти формулы попробуем обосновать через верхний лес Δ
        $\sigma := \varepsilon$
        для каждого узла $N \in \Delta$ с формулой Ψ такой,
                что **унифицировать**(Φ, Ψ, σ)

```
            S := S ∪ {раскрутить(Φ, N, Δ)}
            σ := ε
        конец для
        S := S ∪ {снятие-или(Δ, Φ)}
    конец если
конец если
возврат S
```

Процедура **снятие-или** пытается обосновать формулу Φ через формулы леса Δ по правилу снятия дизъюнкции и возвращает набор подстановок.

```
процедура снятие-или(Δ, Φ)
:: сначала найдем подходящие импликации
пусть I := ∅

пусть σ := ε
для каждого узла N ∈ Δ с формулой Ψ, имеющей вид (Θ imp Ξ)
        и унифицировать(Φ, Ξ, σ)
    I := I ∪ {(подставить-глоб(σ, Θ), N)}
    σ := ε
конец для

S := ∅

:: теперь найдем подходящие дизъюнкции
пусть σ := ε
для каждого узла N ∈ Δ с формулой Ψ, имеющей вид (Θ1 or Θ2) и сущ.
        Ξ1, Ξ2 такие, что унифицировать(Θ1, Ξ1, σ) и
        унифицировать(Θ2, Ξ2, σ) и сущ. такие Υ1, Υ2, N1, N2, что
        (Ξ1 imp Υ1, N1) ∈ I и (Ξ2 imp Υ2, N2) ∈ I
    Ψ' := подставить-глоб(σ, Ψ)
    S1 := раскрутить(Ψ', N, Δ)
    S2 := раскрутить(Ξ1 imp Υ1, N1, Δ)
    S3 := раскрутить(Ξ2 imp Υ2, N2, Δ)
    S := S ∪ комбинировать(комбинировать(S2, S3), S1)
    σ := ε
конец для
возврат S
```

Процедура **снятие-или2** пытается обосновать формулу Φ через формулы леса Δ по правилу снятия дизъюнкции и возвращает набор подстановок.

процедура **снятие-или2**(Δ, Φ)
пусть $S := \varnothing$
для каждого узла $N \in \Delta$ с формулой Ψ такой, что **подставить-глоб**(ε, Ψ)
имеет вид (Θ `or` Ξ)
$S_1 :=$ **решить**(Δ, Θ `imp` Φ)
$S_2 :=$ **решить**(Δ, Ξ `imp` Φ)
$S_3 :=$ **раскрутить**(Θ `or` Ξ, N, Δ)
$S := S \cup$ **комбинировать**(**комбинировать**(S_1, S_2,), S_3)
конец для
возврат S

Процедура **подставить-глоб** осуществляет подстановку σ в формулу Φ и заменяет оставшиеся локальные метапеременные на новые глобальные. Возвращает формулу.

процедура **подставить-глоб**(σ, Φ)
пусть $\Theta :=$ **подставить**(σ, Φ)

пусть $\{x_1, \ldots, x_n\} =$ **lvar**(Θ)
возврат $\Theta \left[\begin{array}{lcl} x_1 & \leftarrow & \text{новая глоб. перем. типа } \mathbf{type}(x_1), \\ x_2 & \leftarrow & \text{новая глоб. перем. типа } \mathbf{type}(x_2), \\ \ldots & & \\ x_n & \leftarrow & \text{новая глоб. перем. типа } \mathbf{type}(x_n) \end{array} \right]$

Примечание. Предметные переменные, функции и глобальные метапеременные имеют в качестве атрибута *уровень*. Уровень приписывается им в момент возникновения и далее не меняется. Все предметные переменные, функции и глобальные метапеременные, входящие в Γ и Φ исходной задачи $\Gamma \vdash \Phi$, имеют один и тот же уровень, равный 0. При введении новой предметной переменной (процедура **декомпозировать**), новой функции (процедура **замкнуть**) или новой глобальной метапеременной, ей присваивается текущее значение счетчика **Уровень**. Изначально **Уровень** инициализируется нулем, и увеличивается на 1 каждый раз после создания новой предметной переменной, функции или глобальной метапеременной. Уровень переменных и функций учитывается в процедуре унификации и служит для того, чтобы не возникало подстановок вместо метапеременных более низкого уровня (возникших раньше) термов, содержащих переменные и функции более высокого уровня (возникших позже).

Список литературы

[1] Вторушин Ю. И., Замятин А. П., Охотников О. А. Верификация и генерирование текстов языков представления дедуктивных знаний CLASS и INT // Научные труды международной научно-практической конференции «СВЯЗЬ-ПРОМ 2006» в рамках III Евро-Азиатского форума «СВЯЗЬ-ПРОМЭКСПО 2006», Екатеринбург: ЗАО «Компания Реал-Медиа», 2006.

[2] Вторушин Ю. И., Замятин А. П., Охотников О. А. Автоматическое доказательство теорем в проектах CLASS и INT // Научные труды международной научно-практической конференции «СВЯЗЬ-ПРОМ 2007» в рамках IV Евро-Азиатского форума «СВЯЗЬ-ПРОМЭКСПО 2007», Екатеринбург: ЗАО «Компания Реал-Медиа», 2007. С. 174–176.

[3] Вторушин Ю. И., Замятин А. П., Охотников О. А. Автоматическое решение эквациональных задач в САД CLASS и INT // Научные труды международной научно-практической конференции «СВЯЗЬ-ПРОМ 2008» в рамках V Евро-Азиатского форума «СВЯЗЬ-ПРОМЭКСПО 2008», Екатеринбург: ЗАО «Компания Реал-Медиа», 2008.

[4] Bancerek G. On the structure of Mizar types // Electronic Notes in Theoretical Computer Science. 85. 2003. P. 69–85.

[5] Sieg W., Byrnes J. Normal natural deduction proof (in classical logic) // Studia Logica. 60 (1). 1998. P. 67–106.

[6] Sieg W., Cittadini S. Normal natural deduction proof (in non-classical logic) // Mechanizing Mathematical Resoning. LNAI 2605. 2005. P. 169–191.

[7] Trybulec A., Wiedijk F. CHECKER — notes on the basic inference step in Mizar. 2000 /
available at `http://www.cs.kun.nl/~freek/mizar/by.dvi`.