Записки научных
семинаров ПОМИ
Том 293, 2002 г.

Б. Ю. Конев, Т. Жебелеан

МЕТОД ПОДЪЕМА РЕШЕНИЙ ДЛЯ РАБОТЫ С МЕТАПЕРЕМЕННЫМИ В СИСТЕМЕ TH∃OREM∀

1. Введение

В последнее время возобновился интерес к задаче автоматического порождения доказательств, которые могли бы читать не только эксперты в области автоматического доказательства теорем (см., например, [8, 14, 6, 24]). Одним из основных преимуществ поиска "естественно выглядящих" доказательств является то, что в подобные процедуры можно интегрировать методы, используемые математиками в работе, а также применять методы, специфические для различных областей математики, формализованные в виде специальных правил вывода.

Ограничения на вид полученного доказательства, а также необходимость обеспечить возможность динамического расширения процедуры поиска вывода специфическими правилами и разрешающими процедурами для конкретных областей, часто представленными в виде "черных ящиков", затрудняет использование известных эффективных методов таких, как резолюция, табличный метод и метод связок (см., например, обзор [18]). Поэтому часто приходится применять исчисления секвенциального типа, чьим основным недостатком является недетерминизм в выборе собственных термов некоторых кванторных правил (введения $\forall$ в посылки и $\exists$ в заключение). В исчислении "естественного вида" аналогом таким правилам служит поиск "свидетелей" для кванторных правил, особенно для экзистенциональных заключений.

В доказательствах, записанных человеком, подобные термы зачастую обозначаются переменной на мета-уровне, затем зна-

При частичной поддержке гранта #1 6-го конкурса-экспертизы научных проектов молодых ученых РАН (1999). The research described in this publication was made possible in part by Award No. RM1-2409-ST-02 of the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF).
Supported by Austrian Science Foundation (FWF), SFB project F1302.

чения этих метапеременных уточняются на основе полученной в дальнейшем информации. В теории автоматического доказательства теорем такой подход получил название метода *метапеременных* (также называемых *параметрами* или *свободными переменными* – см. [26, 28, 21, 10, 9, 17]) и используется почти во всех системах автоматического поиска доказательств для исчислений секвенциального и табличного типа начиная с 80-х годов XX века (см. [27]).

Любая реализация метода метапеременных состоит из двух фаз: сначала строится (частичное) дерево доказательства, содержащее метапеременные, называемое *преддоказательством* (в литературе также встречается термин *скелет доказательства*). Затем преддоказательство превращается в доказательство подстановкой конкретных термов вместо метапеременных. Грубо говоря, поиск доказательства распадается на пропозициональную часть и удачный выбор нужных термов. Для логики предикатов первого порядка значения метапеременных можно определить при помощи унификации, но задача удачного выбора все равно остается. Можно показать, что поиск корректной подстановки для данного преддоказательства является NP-трудным.

Данная статья имеет две цели. Нашей первой целью является описание нового метода работы с метапеременными, который мы называем *методом подъема решений*. Вместо широко известного перебора возможных деревьев вывода, мы допускаем `AND`- и `OR`-узлы в само дерево вывода. Каждому узлу мы ставим в соответствие подстановку, которая отображает соответствующее этому узлу поддерево в настоящее доказательство. Затем мы "поднимаем" подстановки снизу вверх. Если для некоторого узла подстановки, назначенные его потомкам, не являются совместимыми, преддоказательство необходимо расширить (и в каком-то листе, потомке данного узла). Если нам удается поднять подстановку до корня дерева, поиск доказательства (успешно) завершается.

Полученная процедура делает только ограниченные возвраты: мы никогда не удаляем узлы из дерева вывода, вместо этого перебираются различные подстановки. Перебор подстановок неизбежен при условии что $P \neq NP$.

Вторая цель данной статьи состоит в том, чтобы дать представление о реализации данного метода в системе TH∃OREM∀ [11, 14, 12]. Наша реализация использует механизмы системы, изначально применявшиеся для комбинирования различных методов поиска доказательств и специфических алгоритмов решения задач, добавляя к ним функциональность, необходимую для работы с метапеременными.

Система TH∃OREM∀ является результатом работы группы TH∃OREM∀ group (см. `http://www.theorema.org`) и базируется на более ранних работах Бухбергера [11]. Метод метапеременных был реализован первым автором статьи во время его визита в RISC.

Данная работа является расширенной версией [22].

2. Метапеременные и естественные выводы

Этот раздел посвящен применению метапеременных к поиску собственных термов для исчислений секвенциального типа. Основной вклад данной работы состоит не в том факте, что мы используем метапеременные, а в том способе работы с ними, что представлен в разделе 3. Понятия метапеременной, унификации, сколемовских констант, "ослабленной" сколемизации и различных вариантов упорядочивающих ограничений хорошо известны и могут быть найдены в литературе, например, [1, 16, 17, 25, 23, 2]; однако мы приводим их здесь для того, чтобы избежать ссылок на базовые понятия.

Мы рассматриваем исчисление, которое мы называем *естественным выводом*, основным объектом которого являются *секвенции* с единственной формулой в заключении (сукцеденте) вида

$$A_1, A_2, \ldots, A_n \vdash G \quad \text{или} \quad \Gamma, A \vdash G,$$

где $A_1, A_2, \ldots, A_n$, A обозначают *посылки*, Γ – несколько посылок, а G – заключение (которое надо доказать).

Чтобы построить доказательство секвенции, мы рекурсивно преобразуем ее к одной или нескольким секвенциям до тех пор, пока не преобразуем к множеству очевидно доказуемых секвенций (*заключительных секвенций*). Преобразование осуществляется при помощи правил вывода вида

$$\frac{S_1; \quad S_2; \quad \ldots; \quad S_n}{S}$$

(секвенция S преобразуется к секвенциям $S_1, \ldots, S_n$). Обычно для целей автоматического поиска доказательства (и в этой статье) рассматриваются такие правила вывода, что S выводима тогда и только тогда когда все S_i выводимы, см. раздел 3.1. *Доказательством* секвенции называется дерево секвенций, чей корень помечен начальной (данной) секвенцией, листья помечены заключительными секвенциями, и все секвенции, не находящиеся в листьях, получаются применением какого-то правила вывода к своим потомкам.

"Сложными" правилами являются правила введения кванторов (см. [18]):

$$\frac{\Gamma \vdash F_{\{x \to t\}} \vee \exists x F}{\Gamma \vdash \exists x F}, \quad \frac{\Gamma, F_{\{x \to t\}} \wedge \forall x F \vdash G}{\Gamma, \forall x F \vdash G}, \tag{1}$$

где t является основным термом, а $F_{\{x \to t\}}$ обозначает результат замены всех свободных вхождений переменной x на терм t.

Для того, чтобы применить какое-то из этих правил, очевидно, требуется подобрать подходящий терм t. Основная идея метода метепеременных состоит в том, чтобы отложить построение терма t до тех пор, пока накопившаяся информация позволит определить его значение. Это достигается заменой t в (1) на новую метапеременную ξ, чье значение нам предстоит определить.

$$\frac{\Gamma \vdash F_{\{x \to \xi\}} \vee \exists x F}{\Gamma \vdash \exists x F}, \quad \frac{\Gamma, F_{\{x \to \xi\}} \wedge \forall x F \vdash G}{\Gamma, \forall x F \vdash G}. \tag{2}$$

Доказательство, получаемое с использованием правил (2), называется *преддоказательством*, которое является *корректным*, если существует *корректная* подстановка термов вместо метапеременных, отображющая преддоказательство в настоящее доказательство. (Мы предполагаем, что такая подстановка является основной, т.к. мы можем заменить "нерешенные" метапеременные на новые константы.) Подстановка является корректной, если **(а)** все промежуточные применения правил вывода остаются корректными; **(б)** заключительные секвенции остаются таковыми.

Что касается (а), некорректными могут стать только правила вида

$$\frac{\Gamma \vdash F_{\{x \to c\}}}{\Gamma \vdash \forall x F}, \quad \frac{\Gamma, F_{\{x \to c\}} \vdash G}{\Gamma, \exists x F \vdash G},$$

т.к. в классической формулировке секвенциального исчисления [18] требуется чтобы c была новой свободной переменной (мы также можем считать, что c является константой, такая замена не портит вывода), т.е.

$$c \text{ не входит в } \Gamma,\ F, \text{ и } G.$$

Простой способ обеспечить (4) состоит в том, чтобы ввести ограничения вида $c \notin \xi$ для всех ξ входящих в Γ, F, G, см. [25, 23]. Однако это достаточно сильное требование, которое может приводить к большому количеству откатов и/или большим доказательствам [2].

Мы используем ослабленную версию ограничений, которые не обеспечивают (4), но все же дают корректную процедуру. Для каждой метапеременной ξ и для каждой сколемовской константы c, входящей в F, мы храним ограничения вида

$$c_1 \succ \xi, \ldots, c_n \succ \xi \text{ и } c_1 \succ c, \ldots, c_n \succ c.$$

Чтобы проверить, удовлетворяет ли подстановка σ множеству ограничений R, мы сначала для каждой замены в σ вида $\xi \to t$ (где ξ – метапеременная, t – терм), добавляем ко множеству ограничений ограничения вида $\xi \succ c$ и $\xi \succ \zeta$ для каждой сколемовской константы c и для каждой метапеременной ζ, входящих в t. Затем мы проверяем, что отношение $\succ$ не имеет циклов.

Такой механизм обеспечивает корректность исчисления, поскольку он, в сущности, симулирует метод, предложенный в [2]. Использованные нами сколемовские константы соответствуют сколемовским функциям; ограничения соответствуют аргументам сколемовских функций. Заметим, что отношение $\succ$ также используется для улучшения представления порожденных доказательств, как описано в разделе 4.

Что касается (б), заключительные секвенции имеют вид

$$\Gamma,\ F \vdash F, \quad \text{и} \quad \Gamma,\ A, \neg A \vdash G,$$

где F и A атомарные формулы. Лист преддоказательства отобразим в заключительную секвенцию тогда и только тогда, когда найдется посылка, унифицируемая с заключением, или пара таких, посылок, что одна из них унифицируется с отрицанием другой. Таким образом, задача о нахождении корректной подстановки сводится к задаче унификации.

3. Метод поиска доказательств

В нашей системе и поиск вывода и полученное доказательство представляется в виде AND/OR-*дерева вывода*. AND-узлы соответствуют обычным применениям правила вывода (см. раздел 2), а OR-узлы представляют альтернативные способы поиска доказательства; OR-узлы не могут быть унарными.

Формально говоря, дерево вывода, порождаемое системой, не является доказательством: в нем содержатся лишние ветви. Мы называем дерево вывода AND/OR-*доказательством*, если для каждого OR-узла найдется наследник такой, что дерево, полученное рекурсивной заменой OR-узла на выбранного наследника, является доказательством, как определено в разделе 2.

Система TH∃OREM∀ комбинирует различные машины вывода, называемые *базовыми процедурами*, некоторые из которых специализированы для работы со специфическими задачами математических теорий. Базовые процедуры возвращают поддерево, представляющее один или несколько шагов вывода. *Управляющая процедура* вызывает базовые процедуры и комбинирует результат их работы в дерево вывода. Детальное описание процедуры поиска вывода в системе TH∃OREM∀ может быть найдено в [15].

Каждой вершине дерева вывода присваивается *статус*. Статус вершины может быть *proved, failed* или *pending* (с очевидным смыслом: доказано, неудача или отложено соответственно). После применения правила вывода, статус некоторых вершин может измениться. Управляющая процедура *обновляет* статус вершин дерева вывода снизу вверх, превращая *pending* в *proved* или *failed*, если это возможно.

Механизм работы с метапеременными распадается на два уровня – уровень базовой процедуры и уровень управляющей процедуры. Базовая процедура *назначает* некоторые значения метапеременным; управляющая процедура обеспечивает *согласованность* значений метапеременных на разных ветках дерева вывода.

3.1. Базовая процедура для логики первого порядка.

В система TH∃OREM∀ базовая процедура для логики первого порядка называется PND (Proof by Natural Deduction) [13]. В данном разделе мы представляем расширение этой процедуры для работы с метапеременными. Процедура реализована как (упорядоченное) множество правил переписывания, соответствующих секвенциальному исчислению.

Первым правилом является правило определения заключительных секвенций. Для каждой секвенции вида

$$\Gamma,\ A \vdash B, \quad (\text{или} \quad \Gamma,\ A, \neg B \vdash G),$$

где A и B – пара бескванторных формул, не использованная раньше, проверяется, унифицируемы ли A и B. Если их наиболее общий унификатор удовлетворяет ограничениям R, поставленным в соответствие узлу дерева вывода, как объясняется далее, процедура поиска создает `OR` узел с двумя ветвями:

- *"доказанная"* ветвь, состоящая из узла, статус которого *proved* и которому поставлены в соответствие новая подстановка и множество ограничений;
- *"отложенная"* ветвь с той же самой секвенцией, но для которой данная подстановка запрещена на будущее.

(Информация о запрещенной подстановке рекурсивно передается наследникам узла.)

Говоря неформально, процедура поиска "пробует" подстановку и оставляет возможность идти далее, если эта подстановка окажется некорректной.

Если данная секвенция не может быть отображена в заключительную некоторой подстановкой (или подстановка несовместима с ограничениями), `PND` пытается применить другие правила, преобразующие посылки или заключение естественным образом.

- Упрощающие и расщепляющие правила:

$$\frac{\Gamma, \neg G_2, \dots, \neg G_k \vdash G_1}{\Gamma \vdash G_1 \vee \dots \vee G_2}; \quad \frac{\Gamma, G_1 \vdash G_2}{\Gamma \vdash G_1 \implies G_2};$$

$$\frac{\Gamma, \neg A \vee B \vdash G}{\Gamma, A \implies B \vdash G}; \quad \frac{\Gamma, A_1, \dots, A_k \vdash G}{\Gamma, A_1 \wedge \dots \wedge A_k \vdash G};$$

$$\frac{\Gamma, G \vdash \text{false}}{\Gamma \vdash \neg G}; \quad \frac{\Gamma, A \vdash G}{\Gamma, \neg\neg A \vdash G}; \quad \frac{\Gamma, \neg A_1 \vee \dots \vee \neg A_k \vdash G}{\Gamma, \neg(A_1 \wedge \dots \wedge A_k) \vdash G};$$

$$\frac{\Gamma, \neg A_1 \wedge \dots \wedge \neg A_k \vdash G}{\Gamma, \neg(A_1 \vee \dots \vee A_k) \vdash G}; \quad \frac{\Gamma, A \wedge \neg B \vdash G}{\Gamma, \neg(A \implies B) \vdash G};$$

$$\frac{\Gamma, \forall x_1, \dots, x_n \neg F \vdash G}{\Gamma, \neg(\exists x_1, \dots, x_n F) \vdash G}; \quad \frac{\Gamma, \exists x_1, \dots, x_n \neg F \vdash G}{\Gamma, \neg(\forall x_1, \dots, x_n F) \vdash G}.$$

$$\frac{\Gamma, A_1 \vdash G; \cdots \Gamma, A_k \vdash G}{\Gamma, A_1 \vee \cdots \vee A_k \vdash G}; \quad \frac{\Gamma \vdash G_1; \cdots \Gamma \vdash G_k}{\Gamma \vdash G_1 \wedge \cdots \wedge G_k}.$$

Особый интерес для метода метапеременных представляют следующие

- Правила введения констант

$$\frac{\Gamma \vdash F_{\{x_1 \to c_1, \ldots, x_n \to c_n\}}}{\Gamma \vdash \forall x_1, \ldots, x_n F}, \quad \frac{\Gamma, F_{\{x_1 \to c_1, \ldots, x_n \to c_n\}} \vdash G}{\Gamma, \exists x_1, \ldots, x_n F \vdash G},$$

где $c_1, \ldots c_n$ новые сколемовские константы[1]. Для каждой метапеременной x и каждой сколемовской константы c, входящих в F, мы ставим в соответствие узлу дерева вывода набор ограничений вида

$$c_1 \succ \xi, \ldots, c_n \succ \xi \text{ и } c_1 \succ c, \ldots, c_n \succ c,$$

как объясняется в разделе 2; эти ограничения рекурсивно передаются потомкам узла.

- Правила введения метапеременных:

$$\frac{\Gamma \vdash G_{\{x_1 \to \xi_1, \ldots, x_n \to \xi_n\}} \vee \exists x_1, \ldots, x_n G}{\Gamma \vdash \exists x_1, \ldots, x_n G};$$

$$\frac{\Gamma, A_{\{x_1 \to \xi_1, \ldots, x_n \to \xi_n\}} \wedge \forall x_1, \ldots, x_n A \vdash G}{\Gamma, \forall x_1, \ldots, x_n A \vdash G},$$

где $\xi_1, \ldots, \xi_n$ – новые метапеременные.

3.2. Управляющая процедура.

Рассмотрим, для начала, случай детерминированного хода поиска доказательства, то есть предположим, что дерево вывода содержит только AND-узлы. В процессе построения преддоказательства, каждому узлу ставится в соответствие множество ограничений R вида $a \succ b$, которое затем рекурсивно передается сверху вниз. Каждому заключительному узлы назначаются статус *proved* и подстановка, делающая его заключительным узлом. Если все потомки AND-узла доказаны, т.е., имеют статус *proved*, мы пытаемся скомбинировать соответствующие подстановки и

[1] В классической формулировке секвенциального исчисления [18] вместо констант используются свободные переменные. Замена этих свободных переменных на константы оставляет доказательство верным и выглядит более естественно для непрофессионалов.

ограничения как показано ниже (совместимость подстановок и ограничений проверяется как описано в предыдущем разделе). Если такое комбинирование удается, узел также становится доказанным. Действуя по индукции, процедура закончит работу, породив подстановку, корректную для узла дерева. После этого нетронутые метапеременные заменяются на новые константы, чтобы сделать вывод основным.

Описанная процедура осуществляется при помощи функции
Function Simple_Combine

Вход: Набор пар $\{(\lambda_1, R_1), (\lambda_2, R_2), \ldots, (\lambda_n, R_n)\}$ [подстановки и ограничения потомков].

Выход: Пара (σ_N, R_N) или `NULL` [подстановки и ограничения для родительского узла].

Метод:
- Пусть $\lambda_i = \{\xi_{i,k} \to t_{i,k}\}_{k=1,\ldots}$.
- Найти наиболее общий унификатор σ_N для $\langle \xi_{1,1} \ldots \xi_{n,1} \ldots \rangle$ и $\langle t_{1,1} \ldots t_{n,1} \ldots \rangle$.
- Если σ_N существует и удовлетворяет $\cup_i R_i$, вернуть $(\sigma_N, \cup_i R_i)$ иначе вернуть `NULL`.

Очевидно, что итоговая подстановка λ является наиболее общим унификатором, отображающим соответствующее поддерево в верное доказательство. (Заметим, что среди наших упорядочивающих ограничений не встречаются упорядочивания, где слева и справа от $\succ$ стоят метапеременные; наиболее общие унификаторы могут различаться лишь переименованием метапеременных, поэтому, любой наиболее общий унификатор может быть выбран в данной функции.)

Заметим далее, что применяемый алгоритм унификации может быть приспособлен для решения задач специальных областей, если подстановку можно строить последовательно, т.е. по данным решениям для потомков $N_1, N_2, \ldots, N_n$ некоторого узла, должна быть возможность построить решение, которое *одновременно* удовлетворяет всем этим узлам, либо показать, что такого решения нет.

Рассмотрим случай, когда дерево вывода содержит `OR`-узлы. Среди потомков некоторого `OR`-узла N могут встречаться *pending* потомки и *proved* потомки $N_1, \ldots, N_n$, которым поставлены в соответствие пары $(\lambda_1, R_1), \ldots, (\lambda_n, R_n)$. Естественно назначить N множество $\{(\lambda_1, R_1), \ldots, (\lambda_n, R_n)\}$ “возможных” решений.

pending потомки не имеют назначенных решений, мы храним список указателей на эти вершины, чтобы иметь возможность продолжить поиск доказательства в случае неудачи.

Теперь мы готовы описать общий случай. Каждому узлу мы ставим в соответствие *систему решений* – множество, чьими элементами являются либо пары вида (σ_N, R_N), либо указатели на узлы дерева вывода. Пусть функции `Pairs`() и `Pointers`() выбирают из системы решений элементы первого и второго типов соответственно. Узел является доказанным, если назначенная ему система решений содержит пару. Таким образом, системы решений играют ключевую роль в вычислении статуса вершин, в действительности, мы вычисляем статус и строим системы решений при помощи одной функции `Compute_Value`.

Чтобы описать функцию `Compute_Value`, нам потребуется две дополнительные функции `CombineOR` и `CombineAND`, вычисляющие системы решений для `OR`- и `AND`-узлов дерева вывода.

Система решений `OR`-узла представляет собой объединение систем решений его непосредственных потомков.

Function `CombineOR`

Вход: Набор систем решений $\{S_1, \ldots, S_r\}$.

Выход: Система решений S.

Метод:

- Вернуть объединение $S_1, \ldots, S_r$.

Если все непосредственные потомки `AND`-узла доказаны, каждому из них поставлена в соответствие некоторая система решений, содержащая пару. Обозначим эти системы $S_1, \ldots, S_r$. Каждая подстановка, входящая в `Pairs`(S_i) является корректной для соответствующего поддерева. `CombineAND` выбирает по элементу s_i из каждого `Pairs`(S_i) для всех $i : 1 \leq i \leq r$ и передает $s_1, \ldots, s_r$ функции `Simple_Combine`. Результат работы `Simple_Combine` либо содержит подстановку, либо есть `NULL`.

Если никакая подстановка не может быть получена таким образом, функция возвращает объединение указателей, входящих в данные системы решений. Процедура поиска доказательства может продолжить построение дерева вывода, начиная с одной из этих вершин, что может привести к появлению новых доказанных вершин.

Если данные системы решений не содержат указателей, доказать данный узел невозможно и функция возвращает `NULL`.

Function `CombineAND`

Вход: Набор систем решений $\{S_1, \ldots, S_r\}$.

Выход: Система решений $\{(\sigma_1, R_1), \ldots, (\sigma_M, R_M), p_1, \ldots, p_N\}$, набор указателей $\{p_1, \ldots, p_N\}$ или `NULL`.

Метод:

- Пусть $S = \emptyset$.
- Пусть $P =$ `POINTERS`$(S_1) \cup \cdots \cup$ `POINTERS`(S_r).
- Для всех наборов $(s_1, s_2, \ldots, s_r) \in$ `Pairs`$(S_1) \times$ `Pairs`$(S_2) \times \cdots \times$ `Pairs`(S_r)
  - если `Simple_Combine`$(s_1, \ldots, s_r)$ $\neq$`NULL`, добавить выход `Simple_Combine`$(s_1, \ldots, s_r)$ к S.
- Если $S = \emptyset$ и $P = \emptyset$, вернуть `NULL`,
- иначе если $S = \emptyset$, вернуть P,
- иначе вернуть $S \cup P$.

Функция `Compute_Value` комбинирует вычисление статуса вершин с построением систем решений. Кроме того, она строит глобальный список вершин `NList`.

Function `Compute_Value`

Вход: Узел N, чьими непосредственными потомками являются $N_1, \ldots, N_t$.

Выход: Статус и система решений, назначенные N.

Побочные эффекты: Изменяет глобальный список `NList`.

Метод:

- Если N является `AND`-узлом
  - Если статус одного из узлов $N_1, \ldots, N_t$ есть *failed*, установить статус N в *failed* и выйти.
  - Если статус одного из узлов $N_1, \ldots, N_t$ есть *pending*, установить статус N в *pending* и выйти.
  - Пусть $S_1, \ldots, S_t$ являют системами решений, поставленными в соответствие $N_1, \ldots, N_t$ соответственно. Пусть $S_{\texttt{AND}} =$`CombineAND`$(S_1, \ldots, S_t)$.

  * Если $\mathtt{Pairs}(S_{\mathtt{AND}}) = \emptyset$ и $\mathtt{POINTERS}(S_{\mathtt{AND}}) = \emptyset$, установить статус N в *failed* и выйти.
  * Если $\mathtt{Pairs}(S_{\mathtt{AND}}) = \emptyset$ и $\mathtt{POINTERS}(S_{\mathtt{AND}}) \neq \emptyset$, установить статус N в *pending*, добавить $\mathtt{POINTERS}(S_{\mathtt{AND}})$ к `NList` и выйти.
  * Если $\mathtt{Pairs}(S_{\mathtt{AND}}) \neq \emptyset$, установить статус N в *proved*, поставить в соответствие N систему решений $S_{\mathtt{AND}}$ и выйти.

- Если N является `OR`-узлом
  - Если статус всех узлов $N_1, \ldots, N_t$ есть *failed*, установить статус N в *failed* и выйти.
  - Если статус всех узлов $N_1, \ldots, N_t$ есть *pending* или *failed*, установить статус N в *pending* и выйти.
  - Если статус одного из $N_1, \ldots, N_t$ есть *proved*,
    * Пусть $N'_{i_1}, \ldots, N'_{i_j}$ будут те узлы из $N_1, \ldots, N_t$, чей статус есть *proved*. Пусть $S_{i_1}, \ldots, S_{i_j}$ будут системы решений, поставленные в соответствие $N'_{i_1}, \ldots, N'_{i_j}$ соответственно.
    * Пусть $N''_{k_1}, \ldots, N''_{k_l}$ будут те узлы из $N_1, \ldots, N_t$, чей статус есть *pending*. Пусть $p_{k_1}, \ldots p_{k_l}$ будут указатели на узлы $N''_{k_1}, \ldots, N''_{k_l}$ соответственно. Пусть T будет системой решений $\{p_{k_1}, \ldots p_{k_l}\}$.
    * Установить статус узла N в *proved*, поставить ему в соответствие систему решений $\mathtt{CombineOR}(S_{i_1}, \ldots, S_{i_j}, T)$ и выйти.

Заметим, что некоторые части системы могут ставить узлам в соответствие пустые системы решений $\mathcal{E} = \{\epsilon, \emptyset\}$, где ϵ – пустая (тождественная) подстановка. Пустая система решений может быть скомбинирована с любой системой решений.

Поиск доказательства осуществляется функцией `Proof_Search` (см. также обсуждение стратегий поиска далее).

Function `Proof_Search`

Вход: Секвенция, которую надо доказать, $\Gamma \vdash G$.

Выход: Если $\Gamma \vdash G$ доказуема, дерево вывода T, представляющее доказательство.

Метод:

- Построить дерево вывода T, состоящее из единственного `AND`-узла, соответствующего данной секвенции.

- Пусть `NList`$= \emptyset$.
- Пока статус корня дерева T есть *pending*, повторять
  - Выбрать (в соответствии со стратегией поиска) лист L дерева T, имеющий статус *pending* такой, что одно из условий выполнено:
    1. ни один из предков L не имеет *proved* статуса;
    2. L – наследник одного из узлов, содержащихся в `NList`.
  - Если ни одно из правил вывода не может быть применено к L, установить статус L в *failed*.
  - Применить (согласно стратегии поиска) одной из правил к L, результат представляется в виде поддерева дерева вывода.
  - Обновить статус вершин дерева, двигаясь от листьев к корню, при помощи функции `Compute_Value`. Обновление статуса может изменить список `NList`.
  - Исключить из списка `NList` все узлы, у которых нет потомков со статусом *pending*.

Заметим, что в `AND/OR`-выводе, построенном описанной функцией поиска, каждый `OR`-узел обладает следующим свойством: один из двух наследников является заключительным узлом, поэтому, в действительности, откаты не осуществляются. Заметим, что описанная процедура не ограничена этим случаем и может применяться к исчислениям, где откаты неизбежны, таким как, например, метод связок [7].

Стратегии поиска вывода. Чтобы функция `Proof_Search` могла найти доказательство, необходимо применять *честную* стратегию вывода. Стратегия является честной, если на недоказуемой секвенции при наличии неограниченных ресурсов для каждой ветви дерева вывода верно следующее: к каждой формуле, встречающейся в какой-либо секвенции данной ветви, некоторое правило вывода будут рано или поздно применено, и если формула является универсальной посылкой или экзистенциональным заключением, метапеременные вводятся в нее бесконечно много раз. (Наше понятие честности стратегии совпадает с понятием, используемым в семантических таблицах, см. [8].)

Стратегия, применяемая в системе, такова: мы расширяем дерево вывода методом поиска в глубину до тех пор, пока не найдем кандидата на то, чтобы быть заключительной секвенцией.

Когда все потомки некоторого **AND**-узла доказаны, мы пытаемся скомбинировать назначенные им системы решений. Если это невозможно, мы расширяем дерево в листе, ближайшем к **AND**-узлу. Таким образом, мы комбинируем поиск в глубину для построения дерева и поиск в ширину для выбора следующего узла в случае неудачи.

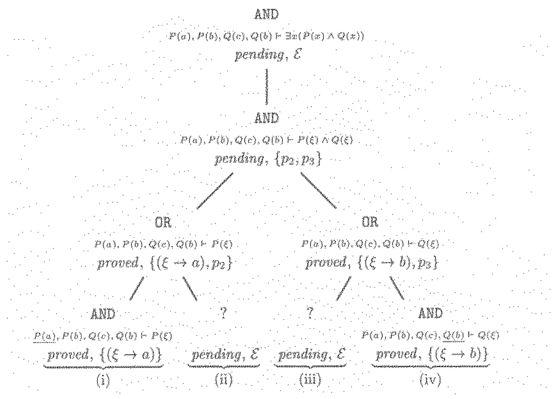

Рис. 1. Дерево вывода для $P(a), P(b), Q(c), Q(b) \vdash \exists x(P(x) \wedge Q(x))$.

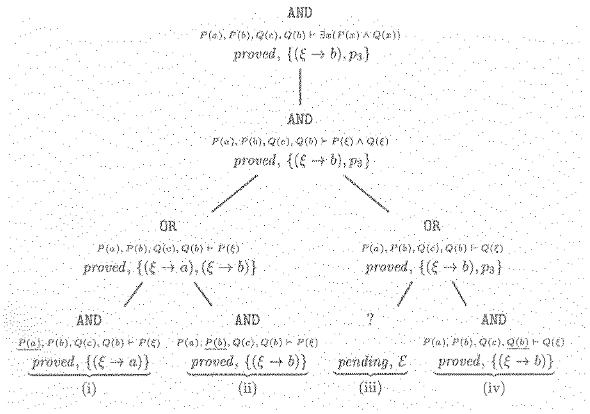

Рис. 2. Законченное дерево вывода для $P(a), P(b), Q(c), Q(b) \vdash \exists x(P(x) \wedge Q(x))$.

На рисунке 1 представлено дерево вывода, на котором показаны статус и системы решений, поставленные в соответствие вершинам дерева. p_2 и p_3 являются указателями на листья (ii) и (iii) соответственно. В момент, представленный на рисунке, список `NList` содержит узлы (ii) и (iii). Поэтому, процедура выбирает в качестве L один из листьев (ii) и (iii), например, лист (ii). Расширенное дерево вывода представлено на рисунке 2.

Система решений, поставленная в соответствие корню дерева, содержит корректную подстановку $\xi \to b$. Указатель p_3 показывает на место, где вывод может быть продолжен.

4. Пример.

В данном разделе мы описываем наши эксперименты с системой на довольно сложном примере. Оказывается, что неограниченное введение метапеременных (как описано в разделе 3.1) приводит к слишком большим и сложным деревьям вывода, поэтому, система, основанная лишь на описанном методе, не может найти доказательство за разумное время. В настоящее время мы работаем над комбинированием данного метода с некоторыми более специальными правилами (такими как инстанциирование заключения благодаря основной посылке), которые применяются до введения метапеременных. Подобный подход особенно хорош для случая, когда секвенции состоят из формул специального вида, например, хорновских формул.

Представленное доказательство было полностью автоматически порождено системой при помощи следующей стратегии:

- Попробовать определить заключительные секвенции.
- Применить упрощающие и расщепляющие правила.
- Поискать соответствие между заключением и основной посылкой (backward reasoning).
- Ввести метапеременные в заключение.
- Поискать соответствие между посылками

Разница между такой стратегией и стратегией, описанной в разделе 3.2, состоит в том, что поиск соответствия между заключением и посылками осуществляется раньше введения метапеременных в заключение и метапеременные вообще не вводятся в посылки. (Выбор последнего пункта зависит от параметров системы.)

В представленном в приложении 1 доказательстве сколемовские константы обозначены буквами с натуральным индексам (например, N_0), тогда как метапеременные обозначены буквами с верхним индексом "$*$" и нижним натуральным индексом (например, N_0^*). Показана только одна успешная ветвь доказательства (упрощения также сделаны автоматически). Удаленные альтернативные ветви содержали дополнительное использование транзитивности (формулы (>=>)) и альтернативные решения для i_0^*, j_0^* (на альтернативное ветви также может быть найдено решение $\max(N_2, N_1)$).

Мы упоминали в разделе 2, что упорядочивающие ограничения имеют практическое значение для представления доказательств, поскольку они могут интерпретироваться следующим образом: если $\xi \succ c$, (где ξ – метапеременная, а c – сколемовская константа), то шаг вывода, вводящий ξ должен быть после шага, вводящего c. Поэтому, $\succ$ указывает порядок, согласно которому шаги вывода должны быть переупорядочены, чтобы в полученном доказательстве условие (4) было верным, что является более естественным.

В представленном выводе, отношение $\succ$ на сколемовских константах и метапеременных таково:

$$n_0 \succ N_0^* \succ i_0^*, j_0^* \succ N_0, N_1 \succ \delta_0 \succ \epsilon_0.$$

Это позволяет переупорядочить шаги вывода, как представлено в приложении 2. (Представлена только часть после строчки, помеченной (9).)

Заметим, что шаги типа "Because (14), we instantiate (eps) to..." были заменены на "We instantiate (eps) to...", чтобы избежать ссылок вперед. В исходном доказательстве система использует (14) как подсказку для (eps), однако эта подсказка не доступна после переупорядочивания. Это дает дополнительное объяснение, почему поиск доказательств с "правильным" порядком более трудоемок, чем с "неправильным".

Доказательство также может быть преобразовано в корректное доказательство (без метапеременных) заменой всех вхождений оставшихся метапеременных на константы и изменением объяснений. Так измененное доказательство представлено в приложении 3.

5. Заключение.

Представленный подход обладает следующими отличительными свойствами:

- Метод может быть использован совместно с любыми правилами вывода, в частности, для работы со специфическими областями математики.
- Метод может быть использован совместно с различными алгоритмами решения и системами переписывания, основанными на системах равенств, при условии, что одновременное решение может быть найдено последовательно.
- Доказательство получается сразу, как только это возможно, несовместимые метаподстановки удаляются сразу, как обнаруживаются.
- Узлы дерева вывода никогда не удаляются, они повторно используются насколько это возможно.
- Метод может быть использован совместно с любой (честной) стратегией построения дерева вывода.
- Метод может быть легко распараллелен.

Похожий подход был независимо открыт М. Гизе [20, 19] для семантических таблиц. Основной целью его работы являлась эффективность. Читаемость полученных доказательств практически не обсуждалась.

Мы кратко опишем другие методы работы с метапеременными, которые нам известны.

В табличных исчислениях со свободными переменными (см., например, [4]), выбор подстановки делается недетерминированно и подстановка немедленно применяется ко всему преддоказательству. Недетерминизм разрешается *перебором* всех возможных подстановок с помощью *возвратов*. Обычно для представления поиска вывода используется `AND`/`OR`-дерево (доказательств), см. [4]. Основным недостатком такой реализации является то, что информация о шагах вывода может быть потеряна при возвратах, поэтому, такие методы называются *деструктивными*.

Недеструктивность метода может быть обеспечена, например, откладыванием применения подстановок до тех пор, пока не будет найдена подстановка, закрывающая все ветви одновременно. Это требует глобальной проверки возможности закрыть дерево после каждого шага вывода.

В последнее время появились полные недеструктивные процедуры, в которых не осуществляются возвраты, [5] (похожие идеи можно найти в [3]). Суть метода состоит в том, что когда подстановка применяется, новое дерево присоединяется ко всем затронутым ветвям дерева вывода. Это дерево “реконструирует” измененные формулы. Таким образом, возвраты избегаются за счет увеличения размера дерева доказательства.

Приложение 1: Автоматически порожденное доказательство

Prove:
(lim(+)) $\lim[f,a] \wedge \lim[g,b] \Rightarrow \lim[f \oplus g, a+b]$,
under the assumptions:
(lim :) $\forall_{a,f}(\lim[f,a] :\Leftrightarrow \forall_{\epsilon}\ (\epsilon > 0 \Rightarrow \exists_N, \forall_n\ (n \geq N \Rightarrow |f[n] - a < \epsilon)))$,
($\oplus$:) $\forall_{f,g,x}((f \oplus g)[x] := f[x] + g[x])$,
($|++|$) $\forall_{a,b,x,y,\delta,\epsilon}(|x-a| < \delta \wedge |y-b| < \epsilon \Rightarrow |(x+y)-(a+b)| < \delta+\epsilon)$,
(max) $\forall_{i,j,k}(k \geq \max[i,j] \Rightarrow k \geq i \wedge k \geq j)$,
(eps) $\forall_{\epsilon}\ (\epsilon > 0 \Rightarrow \exists_{\delta}\ (\delta > 0 \wedge (\delta + \delta = \epsilon)))$,
($<=<$) $\forall_{t,u,v}(u < v \wedge (v = t) \Rightarrow u < t)$.
We prove (lim +) by the deduction rule.
We assume
(1) $\lim[f,a] \wedge \lim[g,b]$
and show
(2) $\lim[f \oplus g, a+b]$.
For proving (2), by the definition (lim :), it suffices to prove:
(3) $\forall_{\epsilon}(\epsilon > 0 \Rightarrow \exists_N \forall_n (n \geq N \Rightarrow \big|(f \oplus g)[n] - (a+b)\big| < \epsilon))$.
Using ($\oplus$:), the goal (3) is transformed into:
(4) $\forall_{\epsilon}(\epsilon > 0 \Rightarrow \exists_N \forall_n (n \geq N \Rightarrow \big|(f[n] + g[n]) - (a+b)\big| < \epsilon))$.
The formula (1.1) is expanded by the definition (lim :) into:
(5) $\forall_{\epsilon}(\epsilon > 0 \Rightarrow \exists_N \forall_n (n \geq N \Rightarrow \big|f[n] - a\big| < \epsilon))$.
The formula (1.2) is expanded by the definition (lim :) into:
(6) $\forall_{\epsilon}(\epsilon > 0 \Rightarrow \exists_N \forall_n (n \geq N \Rightarrow \big|g[n] - b\big| < \epsilon))$.
For proving (4) we take all variables arbitrary but fixed and prove:
(7) $\epsilon_0 > 0 \Rightarrow \exists_N \forall_n (n \geq N \Rightarrow \big|(f[n] + g[n]) - (a+b)\big| < \epsilon_0)$.
We prove (7) by the deduction rule.
We assume
(8) $\epsilon_0 > 0$
and show
(9) $\exists_N \forall_n (n \geq N \Rightarrow \big|(f[n] + g[n]) - (a+b)\big| < \epsilon_0)$.
For proving (9) we have to find N_0^* such that:
(10) $\forall_n (n \geq N_0^* \Rightarrow \big|(f[n] + g[n]) - (a+b)\big| < \epsilon_0)$.
For proving (10) we take all variables arbitrary but fixed and prove:
(11) $n_0 \geq N_0^* \Rightarrow \big|(f[n_0] + g[n_0]) - (a+b)\big| < \epsilon_0$.

We prove (11) by the deduction rule.
We assume
(12) $n_0 \geq N_0^*$
and show
(13) $\left|(f[n_0] + g[n_0]) - (a + b)\right| < \epsilon_0$.
In order to prove (13), by (<=<), it is sufficient to prove:
(14) $\exists_v(\left|(f[n_0] + g[n_0]) - (a + b)\right| < v \wedge (v = \epsilon_0))$.
Because (14), we instantiate (eps) to:
(15) $\epsilon_0 > 0 \Rightarrow \exists_\delta(\delta > 0 \wedge (\delta + \delta = \epsilon_0)$.
From (8) and (15) we obtain by modus ponens
(16) $\exists_\delta(\delta > 0 \wedge (\delta + \delta = \epsilon_0))$.
By (16) we can take appropriate values such that:
(17) $\delta_0 > 0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.
For proving (14) (using (17.2)) it suffices to prove:
(18) $\left|(f[n_0] + g[n_0]) - (a + b)\right| < \delta_0 + \delta_0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.
Because a part of (18) coincides with (17.2), it is sufficient to prove:
(19) $\left|(f[n_0] + g[n_0]) - (a + b)\right| < \delta_0 + \delta_0$.
For proving (19), by $(| + +|)$ it suffices to prove
(20) $\left|f[n_0] - a\right| < \delta_0 \wedge \left|g[n_0] - b\right| < \delta_0$.
We prove the individual conjunctive parts of (20):
Proof of (20.1) $\left|f[n_0] - a\right| < \delta_0$:
Because (20.1), we instantiate (5) to:
(21) $\delta_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow \left|f[n] - a\right| < \delta_0)$.
From (17.1) and (21) we obtain by modus ponens
(22) $\exists_N\ \forall_n\ (n \geq N \Rightarrow \left|f[n] - a\right| < \delta_0)$.
By (22) we can take appropriate values such that:
(23) $\forall_n\ (n \geq N_0 \Rightarrow \left|f[n] - a\right| < \delta_0)$.
In order to prove (20.1), by (23), it is sufficient to prove:
(24) $n_0 \geq N_0$.
In order to prove (24), by (max), it is sufficient to prove:
(29) $\exists_j\ (n_0 \geq \max[N_0, j])$.
For proving (29) we have to find j_0^* such that:
(30) $n_0 \geq \max[N_0, j_0^*]$.
Formula (30) is proved because it is identical to (12) with
the substitution $\{N_0^* \to \max[N_0, j_0^*]\}$.
Proof of (20.2) $\left|g[n_0] - b\right| < \delta_0$:
Because (20.2), we instantiate (6) to:
(25) $\delta_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow \left|g[n] - b\right| < \delta_0)$.
From (17.1) and (25) we obtain by modus ponens

(26) $\exists_N\ \forall_n\ (n \geq N \Rightarrow \left|g[n]-b\right| < \delta_0)$.
By (26) we can take appropriate values such that:
(27) $\forall_n\ (n \geq N_1 \Rightarrow \left|g[n]-b\right| < \delta_0)$.
In order to prove (20.2), by (27), it is sufficient to prove:
(28) $n_0 \geq N_1$.
In order to prove (28), by (max), it is sufficient to prove:
(31) $\exists_i\ (n_0 \geq \max[i, N_1])$.
For proving (31) we have to find i_0^* such that:
(32) $n_0 \geq \max[i_0^*, N_1]$.
Formula (32) is proved because it is identical to (12) with
the substitution $\{N_0^* \to \max[i_0^*, N_1]\}$.
By combining the resulting substitutions we obtain:
$\{\{j_0^* \to N_1,\ i_0^* \to N_0,\ N_0^* \to \max[N_0, N_1]\}\}$.

Приложение 2: Доказательство с переупорядоченными применениями правил вывода

идентичная часть опущена
(9) $\exists_N\ \forall_n\ (n \geq N \Rightarrow \left|(f[n]+g[n])-(a+b)\right| < \epsilon_0)$.
We instantiate (eps) to:
(15) $\epsilon_0 > 0 \Rightarrow \exists_\delta\ (\delta > 0 \wedge (\delta+\delta = \epsilon_0))$.
From (8) and (15) we obtain by modus ponens
(16) $\exists_\delta\ (\delta > 0 \wedge (\delta+\delta = \epsilon_0))$.
By (16) we can take appropriate values such that:
(17) $\delta_0 > 0 \wedge (\delta_0+\delta_0 = \epsilon_0)$.
We instantiate (5) to:
(21) $\delta_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow \left|f[n]-a\right| < \delta_0)$.
From (17.1) and (21) we obtain by modus ponens
(22) $\exists_N\ \forall_n\ (n \geq N \Rightarrow \left|f[n]-a\right| < \delta_0)$.
By (22) we can take appropriate values such that:
(23) $\forall_n\ (n \geq N_0 \Rightarrow \left|f[n]-a\right| < \delta_0)$.
We instantiate (6) to:
(25) $\delta_0 > 0 \Rightarrow \exists_N\ \forall_n\ (n \geq N \Rightarrow \left|g[n]-b\right| < \delta_0)$.
From (17.1) and (25) we obtain by modus ponens
(26) $\exists_N\ \forall_n\ (n \geq N \Rightarrow \left|g[n]-b\right| < \delta_0)$.
By (26) we can take appropriate values such that:
(27) $\forall_n\ (n \geq N_1 \Rightarrow \left|g[n]-b\right| < \delta_0)$.
For proving (9) we have to find N_0^* such that:
(10) $\forall_n\ (n \geq N_0^* \Rightarrow \left|(f[n]+g[n])-(a+b)\right| < \epsilon_0)$.
For proving (10) we take all variables arbitrary but fixed and prove:

(11) $n_0 \geq N_0^* \Rightarrow \left|(f[n_0] + g[n_0]) - (a + b)\right| < \epsilon_0$.
We prove (11) by the deduction rule.
We assume
(12) $n_0 \geq N_0^*$
and show
(13) $\left|(f[n_0] + g[n_0]) - (a + b)\right| < \epsilon_0$.
In order to prove (13), by (<=<), it is sufficient to prove:
(14) $\exists_v \left(\left|(f[n_0] + g[n_0]) - (a + b)\right| < v \wedge (v = \epsilon_0)\right)$.
For proving (14) (using (17.2)) it suffices to prove:
(18) $\left|(f[n_0] + g[n_0]) - (a + b)\right| < \delta_0 + \delta_0 \wedge (\delta_0 + \delta_0 = \epsilon_0)$.
Because a part of (18) coincides with (17.2), it is sufficient to prove:
(19) $\left|(f[n_0] + g[n_0]) - (a + b)\right| < \delta_0 + \delta_0$.
For proving (19), by ($| + +|$) it suffices to prove
(20) $\left|f[n_0] - a\right| < \delta_0 \wedge \left|g[n_0] - b\right| < \delta_0$.
We prove the individual conjunctive parts of (20):
Proof of (20.1) $\left|f[n_0] - a\right| < \delta_0$:
In order to prove (20.1), by (23), it is sufficient to prove:
(24) $n_0 \geq N_0$.
In order to prove (24), by (max), it is sufficient to prove:
(29) $\exists_j \ (n_0 \geq \max[N_0, j])$.
For proving (29) we have to find j_0^* such that:
(30) $n_0 \geq \max[N_0, j_0^*]$.
Formula (30) is proved because it is identical to (12) with
the substitution $\{N_0^* \to \max[N_0, j_0^*]\}$.
Proof of (20.2) $\left|g[n_0] - b\right| < \delta_0$:
In order to prove (20.2), by (27), it is sufficient to prove:
(28) $n_0 \geq N_1$.
In order to prove (28), by (max), it is sufficient to prove:
(31) $\exists_i \ (n_0 \geq \max[i, N_1])$.
For proving (31) we have to find i_0^* such that:
(32) $n_0 \geq \max[i_0^*, N_1]$.
Formula (32) is proved because it is identical to (12) with
the substitution $\{N_0^* \to \max[i_0^*, N_1]\}$.
By combining the resulting substitutions we obtain:
$\{\{j_0^* \to N_1, i_0^* \to N_0, N_0^* \to \max[N_0, N_1]\}\}$.

Приложение 3: Доказательство без метапеременных

идентичная часть опущена
For proving (9) , by taking $\{N \to \max[N_0, N_1]\}$, it suffices to prove:

(10) $\forall_n (n \geq \max[N_0, N_1] \Rightarrow \left|(f[n] + g[n]) - (a + b)\right| < \epsilon_0)$.
For proving (10) we take all variables arbitrary but fixed and prove:
(11) $n_0 \geq \max[N_0, N_1] \Rightarrow \left|(f[n_0] + g[n_0]) - (a + b)\right| < \epsilon_0$.
We prove (11) by the deduction rule.
We assume
(12) $n_0 \geq \max[N_0, N_1]$
идентичная часть опущена
For proving (29), by taking $\{j \rightarrow N_1\}$, it suffices to prove:
(30) $n_0 \geq \max[N_0, N_1]$.
Formula (30) is proved because it is identical to (12).
идентичная часть опущена
For proving (31), by taking $\{i \rightarrow N_0\}$, it suffices to prove:
(32) $n_0 \geq \max[N_0, N_1]$.
Formula (32) is proved because it is identical to (12).

Литература

1. Г. В. Давыдов, С. Ю. Маслов, Г. Е. Минц, В. П. Оревков, анд А. О. Слисенко, *Машинный алгоритм установления выводимости на основе обратного метода*. — Зап. научн. семин. ЛОМИ **16** (1969), 8–15.
2. R. Hähnle and P. Schmitt, *The liberalized δ-rule in free variable semantic tableaux*. — Journal of Automated Reasonin **13**(2) (1994), 211–221.
3. P. Baumgartner, N. Eisinger, and U. Furbach, *A confluent connection calculus*. — CADE99 (1999), 329–343.
4. B. Beckert and R. Hähnle, *Analytic tableaux*. in: Automated Deduction a Bases for Applications W. Bibel and P. Schmitt, eds. **8**, in Applied Logic Series, Kluver Academic Publishers (1998), 11–41.
5. Bernhard Beckert, *Depth-first proof search without backtracking for free variable semantic tableaux*. — in: Position Papers, International Conference on Automated Reasoning with Analytic Tableaux and Related Methods, Saratoga Springs, NY, USA, Neil Murray, editor; Technical Report 99-1. Institute for Programming and Logics, Department of Computer Science, University at Albany–SUNY (1999), pp. 33–54.
6. M. Beeson, *Automatic derivation of the irrationality of e*. — CALCULEMUS 99, Electronic Notes in Theoretical Computer Science 99; `www.elsevier.nl/locate/entcs`.
7. W. Bibel, *Automated Theorem Proving*. — Vieweg, Braunschweig, second edition (1987).
8. W. Bibel and P. Schmitt, eds, *Automated Deduction a Bases for Applications*. — **1–3** of Applied Logic Series, Kluver Academic Publishers (1998).
9. K. Broda, *The relationship between semantic tableau and resolution theorem proving*. in: Proceedings of Workshop on Logic, Debrecen, Hungary (1980). Also published as technical report, Imperial College, Department of Computing, London, UK.

10. F. M. Brown, *Toward the automation of set theory and its logic.* — Artificial Intelligence **10** 1978, 281–316.
11. B. Buchberger, *Using* **Mathematica** *for doing simple mathematical proofs.* in: 4th International *Mathematica* users' conference, Tokyo. Wolfram Media Publishing (1996).
12. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, and W. Windsteiger, *Theorema: A progress report.* Technical Report 99-42, Research Institute for Symbolic Computation, December (1999).
13. B. Buchberger and T. Jebelean, *The predicate logic prover.* in: Proceedings of the First International Theorema Workshop, number 97-20 in RISC Linz report series. Research Institute for Symbolic Computation, June (1997).
14. B. Buchberger, T. Jebelean, F. Kriftner, M. Marin, E. Tomuta, and D. Vasaru, *A survey of the* **Theorema** *project.* W. Kuechlin, editor, Proceedings of ISSAC'97 (International Symposium on Symbolic and Algebraic Computation), Maui, Hawaii, July (1997).
15. B. Buchberger and E. Tomuta, *Technical report 98-02, Research Institute for Symbolic Computation (1998).*
16. A.I. Degtyarev, A.V. Lyaletski, and M.K. Morokhovets, *Evidence algorithm and sequent logical inference search.* — LPAR 1999 (1999), 44–61.
17. M. Fitting, *First-Order Logic and Automated Theorem Proving.* Springer, second edition (1996).
18. G. Gentzen, *Untersuchungen über das logische schließ.* Mathematische Zeitschrift **39** (1934), 176–210, 405–433.
19. Martin Giese, *Incremental closure of free variable tableaux.* — to appear in LNCS series (2001).
20. Martin Giese, *Proof search without backtracking using instance streams, position paper.* in: Proc. Int. Workshop on First-Order Theorem Proving, St. Andrews, Scotland (2000).
21. C. Kanger, *Computer Programming and Formal Systems/Studies in Logic and Foundations of Mathematics. North-Holland (1963), pp. 87–93.*
22. B. Konev and T. Jebelean, *Using meta-variables for natural deduction in theorema.* M. Kerber and M. Kohlhase, editors, Calculemus 2000: integration of symbolic computation and mechanized reasoning, A. K. Peters, Natik, Massatchussets (2000).
23. J. Krajíček and P. Pudlák, *The number of proof lines and size of proofs in first order logic.* Arch. Math. Logic **27** (1988), 69–84.
24. E. Melis and V. Sorge, *Employing external reasoners in proof planning.* in: CALCULEMUS 99, Electronic Notes in Theoretical Computer Science, 99. `www.elsevier.nl/locate/entcs`.
25. V. P. Orevkov, *Complexity of Proofs and Their Transformations in Axiomatic Theories.* Translations of Mathematical Monographs. American Mathematical Society (1991).
26. D. Prawitz, *An improved proof procedure.* Theoria 26 (1960).
27. A. Robinson and A. Voronkov, *Handbook of Automated Reasoning.* Elsevier (2001).
28. H. Wang, *Toward mechanical mathematics.* IBM J. of Research and Develpment, **4**(1) (1960).

Konev B.Yu. , Jebelean T. Solution lifting method for handling Meta-variables in the TH∃OREM∀ system.

We approach the problem of finding witnessing terms in proofs by the method of meta-variables. We describe an efficient method for handling meta-variables in natural style proofs and its implementation in the TH∃OREM∀ system. The method is based on a special technique for finding meta-substitutions when the proof search is performed in an **AND/OR** deduction tree. The implementation does not depend on the search strategy and allows easy integration with various special provers as well as with special unification/solving engines. We demonstrate the use of this method in the context of a special forward/backward inference strategy for producing proofs in elementary analysis.

С.-Петербургское отделение
Математического института
им. В. А. Стеклова РАН
konev@pdmi.ras.ru
Research Institute for Symbolic
Computation, A-4232 Hagenberg, Austria
tudor@risc.uni-linz.ac.at

Поступило 15 декабря 2002 г.