EESTI NSV TEADUSTE AKADEEMIA TOIMETISED.
FÜÜSIKA * MATEMAATIKA
ИЗВЕСТИЯ АКАДЕМИИ НАУК ЭСТОНСКОЙ ССР. ФИЗИКА * МАТЕМАТИКА
PROCEEDINGS OF THE ACADEMY OF SCIENCES OF THE ESTONIAN SSR.
PHYSICS * MATHEMATICS

1986, 35, 4

УДК 510

Г. МИНЦ

ПОЛНОЕ ИСЧИСЛЕНИЕ ДЛЯ ЧИСТОГО ПРОЛОГА

(Представил Э. Тыугу)

1. Введение

Эта работа возникла в результате попыток переформулировать операционную семантику языка ПРОЛОГ [1], заданную описанием отдельных шагов работы интерпретатора, в дедуктивное описание, близкое по форме к исходному замыслу хорновского программирования [2], согласно которому программа должна почти совпадать со спецификацией задачи на языке исчисления предикатов, точнее — хорновских дизъюнктов.

Мы построим исчисление такое, что выводимость $\vdash A$ в этом исчислении, настроенном на программу P, эквивалентна тому, что программа P с целью A заканчивает работу с положительным результатом. При этом подразумевается полная, а не частичная унификация (т. е. например, термы $f(x, g(x))$ и $f(x, x)$ не унифицируемы), но рассматривается подмножество, где допускаются лишь хорновские дизъюнкты и операция управления / (играющая, по мнению некоторых авторов, роль «неохраняемого» безусловного перехода go to), а также учитывается порядок обработки предложений, принятый в интерпретаторах ПРОЛОГа.

Именно эти два обстоятельства составляют основное (с точки зрения автора) отличие ПРОЛОГа от хорновского программирования. Исполнение хорновской программы можно с большой точностью представить себе как поиск вывода в исчислении резолюций согласно входной единичной стратегии.

Моделирование интерпретатора ПРОЛОГа потребовало явного введения в язык оператора неудачи $\sim$ (failure), означающего, что попытка доказать данную цель или использовать в доказательстве данное предложение приведет к неудаче за конечное число шагов. Оператор $\sim$ используется только в самой внешней позиции, и не допускаются такие возможности, имеющиеся в некоторых версиях ПРОЛОГа, как образование новых атомов с помощью связки NOT, исполняемые предикаты, добавление дизъюнктов в процессе исполнения программы или использование предикатов от предикатов.

Мы постараемся сделать изложение замкнутым, чтобы статью можно было читать без предварительного знакомства с хорновским программированием или ПРОЛОГом, однако некоторое знакомство с методом резолюции все же требуется. В качестве введения в общую часть хорновского программирования и ПРОЛОГа, которое позволит читателю освоиться со структурой и методами написания логических программ, будет дано короткое и, по-видимому, хорошо известное доказательство универсальности логического программирования, а именно, будут выписаны программы вычисления (частично) рекурсивных функций. Затем будет приведено исчисление для программ без оператора управления /, а затем — исчисление для всего рассматриваемого языка.

Автор благодарен А. Ломпу, В. Нейману, М. Розенфельд, К. Урбайтису и Э. Тыугу за полезное обсуждение различных вопросов, связанных с хорновским программированием и ПРОЛОГом.

2. Язык хорновских дизъюнктов

Напомним основные понятия, связанные с методом резолюции и применяемые также в ПРОЛОГе.

В рассматриваемом языке имеются *индивидные переменные*, обозначаемые через u, v, w, x, y, z, u_1, v_1, w_1, x_1, y_1, z_1, ... , *функциональные символы* (сокращенно — функции), обозначаемые через f, g, h, f_1, g_1, h_1, ... , *предикатные символы* (сокращенно — предикаты), обозначаемые через Q, R, P_1, Q_1, R_1, ... Подразумевается, что каждый функциональный и предикатный символ имеет свое фиксированное количество аргументов, в частности, возможны 0-местные функции (т. е. *константы*) и 0-местные предикаты (т. е. *высказывания*).

Термы строятся обычным образом из индивидных переменных с помощью функций. Например, термом будет выражение $f(x, c, g(y, x), h(z, z))$, где f — четырехместная функция, g, h — двухместные функции, c — константа. Термы обозначаются через p, q, r, t, p_1, q_1, r_1, t_1, ...

Атомы — это выражения вида $Q(t_1, \ldots, t_n)$, где Q — n-местный предикат, $t_1, \ldots, t_n$ — термы.

Наконец, *хорновские дизъюнкты* — это выражения вида

$$A \leftarrow B_1, \ldots, B_n \quad \text{и} \quad \leftarrow B_1, \ldots, B_n, \tag{1}$$

где $A, B_1, \ldots, B_n$ — атомы, $n \geqslant 0$. В методе резолюций им соответствуют выражения

$$A \vee \neg B_1 \vee \ldots \vee \neg B_n \quad \text{и} \quad \neg B_1 \vee \ldots \vee \neg B_n. \tag{2}$$

Второй из дизъюнктов (1) мы, опуская стрелку, запишем в виде $B_1, \ldots, B_n$.

В языке ПРОЛОГ в число атомов B_i может входить константный атом /, играющий, по мнению некоторых авторов, роль оператора безусловного перехода go to. Мы проведем изложение для языка без /, а в последнем разделе статьи укажем изменения, которые нужно внести, чтобы учесть /.

Подстановка — это выражение вида $[x_1 := t_1, \ldots, x_n := t_n]$, где x_i — различные переменные, t_i — термы. Если обозначить эту подстановку через s, то результат Es применения этой подстановки к выражению E получается заменой всех вхождений x_1 на t_1, ... , x_n на t_n.

Унификатор выражений E и F — это подстановка s, *унифицирующая* E и F, т. е. такая, что $Es = Fs$. *Наиболее общий унификатор* $MGU(E, F)$ — это наименьший унификатор в том смысле, что из $Es' = Fs'$ следует существование подстановки s'' такой, что $s' = s''$ $MGU(E, F)$. Подстановка, стоящая здесь в правой части, изображает последовательное выполнение $MGU(E, F)$ и s''.

В основе хорновского программирования лежит правило вывода

$$\frac{A' \leftarrow Z; \quad A, X}{(Z, X)s}, \tag{3}$$

где $s = MGU(A', A)$, т. е. s есть наиболее общий унификатор атомов A' и A. Это правило, переходящее при переводе (2) в частный случай правила резолюции, позволяет получить из посылок $A' \leftarrow Z$ и A, X новый дизъюнкт — заключение $(Z, X)s$.

Программа P — это список хорновских дизъюнктов, называемых

предложениями, цель — это дизъюнкт вида $\leftarrow A$, т. е. в нашей записи просто A. Как и в общей схеме метода резолюции получение цели A из предложений P сводится к выводу пустого дизъюнкта из P и A, в нашем случае — по правилу (3).

3. Универсальность хорновского программирования

Проиллюстрируем возможности хорновского программирования, написав программы для произвольных вычислимых функций. Тем самым будет установлена и универсальность (в смысле теории алгоритмов) хорновского языка. Кроме того, в дальнейшем окажется, что особенности стратегии поиска, характеризующие ПРОЛОГ, малосущественны для этих программ, так как для любых двух правил

$$A \leftarrow X, \quad B \leftarrow Y$$

из одной и той же программы, атомы A и B не унифицируемы. Поэтому одновременно устанавливается и универсальность языка ПРОЛОГ, т. е. его принципиальная пригодность для вычисления любой вычислимой функции. Разумеется, это не означает, что такое вычисление всегда будет практически эффективным.

Напомним, что любая *вычислимая функция* на множестве $N = \{0, 1, 2, \ldots\}$ натуральных чисел получается из исходных функций

$$s(x) = x+1; \quad Z(x) = 0; \quad I_i^n(x_1, \ldots, x_n) = x_i$$

с помощью операций подстановки

$$f = [S, g, h_1, \ldots, h_n]: \qquad f(\mathbf{x}) = g(h_1(\mathbf{x}), \ldots, h_n(\mathbf{x})),$$

примитивной рекурсии

$$f = [R, g, h]: \qquad \begin{aligned} &f(\mathbf{x}, 0) = g(\mathbf{x}) \\ &f(\mathbf{x}, y+1) = h(\mathbf{x}, y, f(\mathbf{x}, y)) \end{aligned}$$

и минимизации

$$f = [\min g]: \qquad f(\mathbf{x}) = \min \{y : g(\mathbf{x}, y) = 0\}.$$

Покажем (индукцией по построению вычислимых функций из исходных с помощью подстановки, примитивной рекурсии и минимизации) как для любой вычислимой функции пишется программа на ПРОЛОГе, причем длина программы оценивается многочленом небольшой степени от числа переменных и длины построения.

Программы будут написаны на языке, содержащем единственную константу 0, единственную одноместную функцию s и ряд предикатов (соответствующих при неформальном понимании графикам функций, участвующих в построении данной функции).

Натуральное число n будем изображать термом $s(s \ldots s(0) \ldots)$, где s повторен n раз.

Т е о р е м а. *Для любой вычислимой функции* $f(\mathbf{x})$ *имеется программа* P_f *такая, что равенство* $f(\mathbf{x}) = z$ *для списка* $\mathbf{x}$, z *натуральных чисел эквивалентно успешному получению цели* $F(\mathbf{x}, z)$ *для программы* P_f.

З а м е ч а н и е 1. Под успешным получением цели можно понимать пока существование вывода пустого дизъюнкта из $P_f + F(\mathbf{x}, z)$ по правилу (3) из раздела 2.

З а м е ч а н и е 2. В действительности при запуске программы P_f для цели $F(\mathbf{x}, z)$, где $\mathbf{x}$ — список натуральных чисел и z — переменная, работа завершается успешно тогда и только тогда, когда значение $f(\mathbf{x})$ определено, и в этом случае результирующая подстановка для z дает как раз значение $f(\mathbf{x})$.

Замечание 3. Подразумевается, что каждой вычислимой функции $f(\mathbf{x})$ (точнее, каждому построению с помощью операций S, R, min) сопоставлен свой предикат $F(\mathbf{x}, z)$.

Доказательство. Применим индукцию по построению f. Ниже $x, y, y_1, x_1, \ldots, x_n, z$ — переменные, $\bar{x}$ — список переменных.

1. f — одна из исходных функций.

1.1. $f \equiv s$. $P_f \equiv$

$$S(x, s(x)) \leftarrow$$

1.2. $f \equiv Z$. $P_f \equiv$

$$Z(x, 0) \leftarrow$$

1.3. $f \equiv I_i^n$. $P_f \equiv$

$$I(x_1, \ldots, x_n, x_i) \leftarrow$$

2. $f \equiv [S, g, h_1, \ldots, h_n]$. $P_f \equiv$

$P_g, P_{h_1}, \ldots, P_{h_n}$

$F(\bar{x}, z) \leftarrow H_1(\bar{x}, x_1), \ldots, H_n(\bar{x}, x_n), G(x_1, \ldots, x_n, z)$.

Иными словами, P_f получается последовательным выписыванием программ $P_g, P_{h_1}, \ldots, P_{h_n}$, за которыми следует новое предложение, выражающее правило вычисления функции f: если $h_i(\bar{x}) = x_i$ $(i = 1, \ldots, n)$ и $g(x_1, \ldots, x_n) = z$, то $f(\bar{x}) = z$.

3. $f \equiv [R, g, h]$. $P_f \equiv$

$P_g, P_h,$

$F(\bar{x}, 0, z) \leftarrow G(\bar{x}, z)$

$F(\bar{x}, s(y), z) \leftarrow H(\bar{x}, y, y_1), G(\bar{x}, y, y_1, z)$.

Последние два предложения выражают базис и шаг рекурсии, определяющей функцию f.

4. $f \equiv [\min, g]$. В программе P_f будет использован дополнительный предикат $Q(\bar{x}, z)$, неформально означающий $(\forall y < z)\, g(\bar{x}, y) \neq 0$. Он определяется соотношением

$$Q(\bar{x}, 0) \quad \text{и} \quad Q(\bar{x}, z+1) \leftarrow Q(\bar{x}, z) \,\&\, g(\bar{x}, z) \neq 0.$$

Последнее неравенство в случае, когда $g(\bar{x}, z)$ определено, означает, что $g(\bar{x}, z) > 0$, т. е. $g(\bar{x}, z) = y+1$ для некоторого y.

Наконец, $f(\bar{x}) = z$ означает, что $g(\bar{x}, z) = 0$ и $(\forall y < z)\ g(\bar{x}, y) \neq 0$. Поэтому

$P_f \equiv$

P_g

$Q(\bar{x}, 0) \leftarrow$

$Q(\bar{x}, s(z)) \leftarrow Q(\bar{x}, z), G(\bar{x}, z, s(y))$

$F(\bar{x}, z) \leftarrow Q(\bar{x}, z)\, G(\bar{x}, z, 0)$.

Доказательство закончено.

4. Работа ПРОЛОГ-программы. P-деревья

Как хорошо известно [3], в задаче хорновского программирования (получить пустой дизъюнкт из системы хорновских дизъюнктов) полна *входная стратегия с упорядочением*, когда правило (3) из раздела 2

$$\frac{A' \leftarrow Z; \quad A, X}{(Z, X)s} \qquad (*)$$

понимается буквально, а не с точностью до перестановки атомов справа от стрелки, т. е. «отрезаются» всегда самые левые атомы.

Заметим, что левая посылка $A' \leftarrow Z$ в (*) обязательно одно из предложений программы, так как ни цель, ни какой-либо из дизъюнктов, полученных по правилу (*), не имеют «головы», т. е. атома, расположенного слева от стрелки.

Стратегия поиска, применяемая в ПРОЛОГе, вводит дополнительное ограничение, состоящее, в первом приближении, в том, что левая посылка — самое первое подходящее предложение данной программы. Здесь предложение $A' \leftarrow Z$ называется *подходящим* для списка A, X, если A' и A начинаются с одного и того же предиката. Это ограничение приводит к тому, что программа

$$a \leftarrow a$$
$$a \leftarrow$$

не достигает успеха для цели a, так как порождается бесконечная цепочка применений первого предложения с результатом a, и до второго предложения очередь никогда не доходит.

В действительности это ограничение несколько ослабляется механизмом возвратов, к описанию которого мы сейчас и переходим.

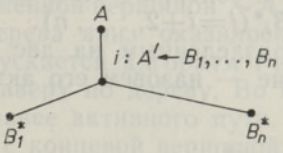
Заметим, что ПРОЛОГ-программа работает по принципу «от цели — к подцели», т. е. заменяет имеющуюся у нее цель совокупностью подцелей с помощью «контрприменения» правила (*). Если в данном состоянии нужно доказать цели из списка A, X и должно применяться предложение $A' \leftarrow Z$, то программа формирует новый список целей $(Z, X)s$, где $s = MGU(A', A)$. При этом в случае успеха цели A, X будут получены с подстановкой s, что мы будем записывать в виде $A, X+s$. Это означает выводимость $\forall x_1 \ldots \forall x_n P \vdash (A \,\&\, X)s$, где $x_1, \ldots, x_n$ — все переменные, входящие в программу P.

В действительности результирующая подстановка для окончательной цели накапливается постепенно, в процессе развертывания по правилу (*) для промежуточных целей.

ПРОЛОГ-программа делает это развертывание путем построения деревьев специального вида, которые мы назовем *P-деревьями*.

Дерево имеет корень (уровня 0). Наследники вершин уровня j имеют уровень $j+1$. В каждой вершине стоит пометка — анализ этой вершины.

$2j$
$2j+1$
$2j+2$



(1)

Анализ вершины четного уровня (*атомарной вершины*) — атомарная формула A и, возможно, некоторая дополнительная информация. Из атомарной вершины исходит не более одной дуги.

Анализ вершины нечетного уровня (*управляющей вершины*), находящейся на дуге, исходящей из атомарной вершины, содержащей атом A — это предложение из программы P

$$i : A' \leftarrow B_1, \ldots, B_n \qquad (2)$$

такое, что атомы A' и A унифицируемы, и, возможно, некоторая дополнительная информация *.

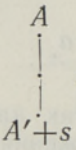
* В (2) подразумевается, что в P все предложения с одним и тем же головным предикатом (т. е. начальным предикатом атома A') перенумерованы. Число i означает номер предложения в этой нумерации.

Если при этом $n>0$, то в атомарных вершинах-наследниках i находятся атомы

$$B_1^*, \ldots, B_n^*, \tag{3}$$

где B_i^* — подстановочный частный случай формулы B_i (какой именно — будет определено позднее).

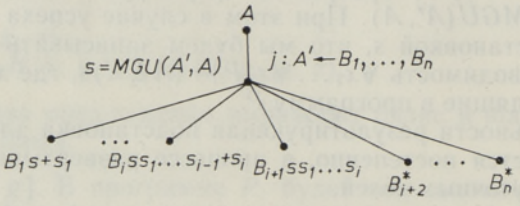
Если $n=0$, то рассматриваемая управляющая вершина i имеет единственного атомарного наследника. В нем находится атом A', знак $+$ (указывающий, что цель, соответствующая данной ветви, достигнута) и подстановка s, где $s=MGU(A', A)$.



(1′)

Сама вершина $A'+s$ наследников не имеет.

Если атомарная вершина помечена знаком $+$, то она содержит наряду с атомом A еще и некоторую подстановку s (что означает достижимость цели A с результирующей подстановкой s). В этом случае все ее братья слева также помечены знаком $+$ и подстановки в (1) для этих $+$-вершин связаны следующим образом.

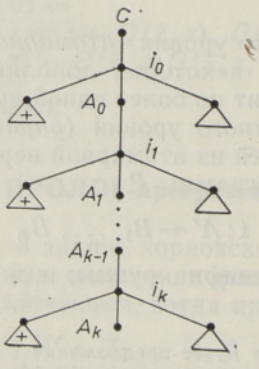


(4)

В действительности $B_j^*=B_j s s_1 \ldots s_{k-1} t$ для $j>i+1$, где t — подстановка, уже накопленная в процессе построения дерева для B^*_{i+1}.

Единственная непомеченная вершина в фигуре (4), которая может иметь наследника, расположена непосредственно справа от самой правой помеченной вершины и содержит атом $B_{i+1} s s_1 \ldots s_i$, а в ее братьях справа находятся атомы B_j^* $(j=i+2, \ldots, n)$.

Дерево оказывается разделенным на две части некоторым путем, начинающимся в его корне — назовем его активным путем. На нашей схеме



(5)

это путь $CA_0A_1 \ldots A_{k-1}A_k$.

Все вершины, расположенные левее активного пути, помечены знаком $+$, остальные вершины не помечены. Концевую вершину активного пути назовем активной вершиной дерева — именно она подлежит «разворачиванию» на очередном шаге поиска.

Опишем теперь *алгоритм* последовательного *развертывания* P-дерева для данной программы P и цели C, состоящий в последовательном построении деревьев $T_1, T_2, \ldots$.

Основной шаг алгоритма — *продолжение дерева*, т. е. дописывание фигуры (1) к некоторой четной вершине A. Работа завершается с положительным результатом, когда дерево оказывается закрытым, т. е. все ветви заканчиваются фигурами (1′).

Иногда продолжение дерева оказывается невозможным. Тогда происходит *возврат*, т. е. отмена одного или нескольких предшествующих шагов, и работа алгоритма продолжается. Если же и возврат невозможен, то работа заканчивается и выдается диагноз «неудача».

Опишем алгоритм NEXT, выдающий по данному дереву следующее за ним дерево. В этом описании участвует вспомогательный алгоритм $\mathrm{NEXT}(A, j)$, описывающий продолжение вершины A фигурой (1) при условии, что первые j подходящих для этого правил уже использованы.

В действительности эти алгоритмы будут применяться и в более общей ситуации, когда одна из атомарных вершин рассматриваемого дерева содержит пометку $\sim$, означающую, что попытка получения соответствующей цели закончилась неудачей за конечное число шагов.

В применении к закрытому дереву для цели C алгоритм NEXT задает поиск следующей (за найденной в данном дереве) подстановки для цели C.

Определим $\mathrm{NEXT}(A, j)$. Если имеется правило с номером $>j$, голова которого унифицируема с A, то пусть i: $A' \leftarrow B_1, \ldots, B_n$ — первое такое правило. В этом случае полагаем для $s = MGU(A', A)$:

$$\mathrm{NEXT}(A, j) = \begin{cases} (1) \text{ при } B_i^* = B_i s \ (i = 1, \ldots, n), \text{ если } n > 0; \\ (1'), \text{ если } n = 0. \end{cases}$$

В противном случае (когда A не унифицируется с головами правил $(j+1), \ldots$) полагаем

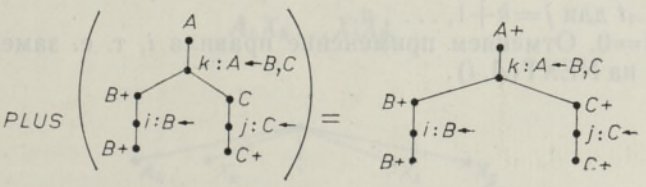
$$\mathrm{NEXT}(A, j) = \sim A,$$

т. е. берем дерево с единственной вершиной $\sim A$.

Если продолжение дерева вниз оказалось удачным и появилась фигура вида (1′), то запускается алгоритм PLUS, продвигающий подстановку s и $+$-отметки вверх по дереву. Во все атомарные вершины, расположенные строго правее активного пути, делается подстановка s. Она считается связанной с концевой вершиной $A'+s$ рассматриваемого пути и будет отменяться, если эта вершина будет выброшена из дерева в процессе возврата (см. ниже).

Если оказалось, что все наследники некоторой непомеченной вершины A активного пути уже помечены знаком $+$, т. е. A входит в фигуру (4), где $i = n$, то A заменяется на $A+ss_1 \ldots s_n$, и эта процедура итерируется.

Например,

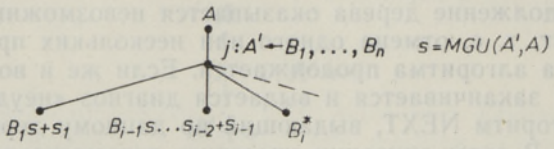


Определим теперь алгоритмы NEXT. Его применение к данному дереву T будет состоять в замене некоторого поддерева на другое с последующим продвижением подстановок и применением алгоритма PLUS по всему полученному дереву. Поэтому мы будем указывать явно только замену. Пусть дано дерево T.

Случай 1. T не содержит $\sim$. Это — шаг продолжения дерева.

1.1. $T=\cdot C$. Заменяем T на NEXT $(C, 0)$.

1.2. T не закрыто. Выбираем в T незакрытую левую атомарную вершину максимального уровня. Это вершина $B_i^*=B_iss_1 \ldots s_{i-1}$ в фигуре



(6)

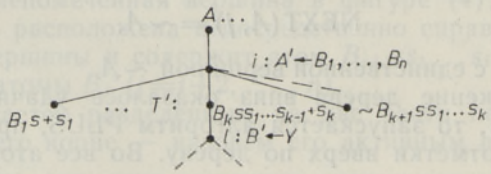
Заменяем вершину B_i^* на дерево NEXT$(B_i, 0)$ и в случае, если это дерево отлично от $\sim$, т. е. начинается ребром, ведущим в управляющую вершину G, делаем унифицирующую подстановку s, соответствующую вершине G, во все атомарные вершины, расположенные правее активного пути. Эта подстановка считается связанной с управляющей вершиной G и отменяется, если G отменяется при возврате (см. ниже).

1.3. T закрыто. Берем в T самую правую ветвь. Она кончается фигурой $(1')$. Отменяем сделанную там подстановку, т. е. заменяем эту часть T на NEXT(A, i) и стираем все знаки $+$ и следующие за ними подстановки на самой правой ветви.

2. Дерево T содержит $\sim$. Это — шаг *возврата*. В этом случае, по нашему соглашению, имеется единственная отметка $\sim$, приписанная активной атомарной вершине.

2.1. $T=\sim A$. Полагаем NEXT$(\cdot \sim A)=\sim A$.

2.2. Вершина, помеченная $\sim$, входит в фигуру



(7)

2.2.1. $k>0$.

Здесь T' обозначает дерево с вершиной $B_kss_1 \ldots s_{k-1}+s_k$. При этом $s_k=tr$, где r — последняя подстановка, сделанная в T', т. е. унификатор из самой правой фигуры вида $(1')$, входящей в это дерево. Дерево NEXT(T) получается путем отмены подстановки r. Точнее, T' заменяется на NEXT(T') и r отменяется во всех атомарных вершинах, находящихся справа от пути, ведущего в вершину $B_kss_1 \ldots s_{k-1}$.

В частности, в братьях этой вершины размещаются формулы $B_jss_1 \ldots s_{k-1}t$ для $j=k+1, \ldots, n$.

2.2.2. $k=0$. Отменяем применение правила i, т. е. заменяем дерево с корнем A на NEXT(A, i).

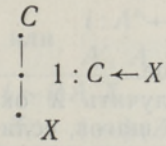
5. Глобальное описание работы ПРОЛОГ-программы и теорема о полноте для языка без /

Опишем структуру «успешного» P-дерева, т. е. дерева, на котором ПРОЛОГ-программа заканчивает работу, когда цель успешно получена. Хотя в нашей исходной постановке целью был один атом, промежуточные подцели, как правило, — списки атомов, а синтаксис ПРОЛОГа допускает список и в качестве первоначальной цели. Поэтому в дальнейшем и мы допускаем такие цели, что и отражено ниже в записи (2). Здесь целью является YA_0X_0, где Y — список правых братьев вершины A_0 (расположенных в $+$-части), а корень дерева считается имеющим уровень -1.

Более формально можно считать, что в рассматриваемую программу добавляется предложение

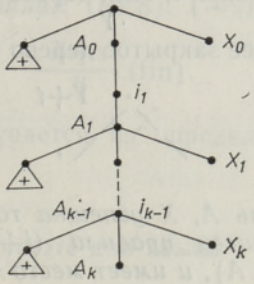
$$1\colon\ C \leftarrow X,$$

где X — рассматриваемая цель, C — новый атом, и ставится задача — получить цель C. Этому соответствует дерево



(1)

Мы не будем использовать этот прием, так что в общем случае P-дерево имеет вид



(2)

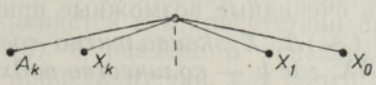
Скажем, что это дерево *успешно*, если P-программа, начиная с него, закончит работу успешно, т. е. на закрытом дереве, причем правило i_{k-1} (а тем самым и каждое из правил i_0, i_1, ...) не будет отменено в процессе работы, так что не будет отменена ни одна из подстановок, уже сделанных левее активного пути A_0, A_1, ... , A_{k-1}, A_k.

В противном случае (т. е. когда правило i_{k-1} отменяется в процессе обработки дерева (2)) скажем, что дерево (2) — отменяемое. В дальнейшем часто используется без прямого упоминания следующее утверждение.

Предложение 1. *Дерево* (2) *успешно тогда и только тогда, когда успешно дерево, состоящее только из цели*

$$A_kX_k \ldots X_1X_0, \tag{3}$$

т. е. дерево



(4)

При этом совпадают как процесс работы, так и результирующие подстановки (с точностью до переименования переменных).

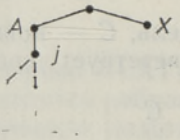
Доказательство проводится индукцией по процессу работы программы, т. е. по развертыванию последовательных P-деревьев.

Введем сокращения для некоторых утверждений о результатах работы ПРОЛОГ-программы.

$Y+s$ — Цель Y успешно получается с результирующей подстановкой s.

$(\sim)Y$ — Попытка получить цель Y заканчивается неудачей за конечное число шагов.

$(\sim j)Y$ — Попытка получить цель Y заканчивается неудачей за конечное число шагов, если требуется, чтобы процесс построения дерева начинался с i-го подходящего (для первого атома из Y) правила, т. е. $Y=A, X$ и дерево



(5_j)

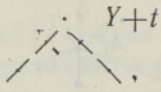
— отменяемое.

$[\sim i]X$ означает $\&_{j\leqslant i}(\sim j)X$ — Попытка получить X оканчивается неудачей за конечное число шагов, если она начинается с любого из первых i подходящих правил.

Следующие утверждения описывают работу ПРОЛОГ-программы, которую мы считаем фиксированной. Скажем, что цель Y успешна с результирующей подстановкой t, если дерево

$$\cdot Y \qquad (6)$$

успешно, и результирующее закрытое дерево имеет вид



т. е. если верно $Y+t$.

Предложение 2. *Цель A, X успешна тогда и только тогда, когда для некоторого подходящего правила $(i+1)$: $A'\leftarrow Z$ успешна цель $(Z, X)s$, где $s=MGU(A', A)$, и имеет место $[\sim i]A$, X, т. е. все деревья вида (5_j), $j\leqslant i$, отменяемы.*

При этом результирующая подстановка для A, X имеет вид ss', где s' — результирующая подстановка для $(ZX)s$.

Схематически это можно записать в виде

$$\frac{(i+1)\colon\quad A'\leftarrow Z;\qquad (ZX)s+s';\qquad [\sim i]A, X}{A, X+ss'}\ (\mathrm{RES})$$

с очевидной модификацией для случая пустого списка Z.

Доказательство. Рассмотрим P-дерево для цели A, X. При его обработке ПРОЛОГ-программа последовательно строит деревья (5_j) до тех пор, пока не будет получено успешное, номер которого мы обозначим через $i+1$. Все предыдущие деревья (5_j) оказались тем самым неуспешными, что дает нам утверждение $[\sim i]A, X$ — правую посылку правила (RES). Оставшаяся часть предложения 2 получается, если вспомнить алгоритм NEXT(A, j) «разворачивания» дерева. □

Рассмотрим теперь очевидные возможные причины неудачи.

Предложение 3. *$(\sim)A$, X эквивалентно совокупности $(\sim 1)A$, X; $(\sim 2)AX$; ... ; $(\sim k)AX$, где k — количество подходящих правил для A.*

Доказательство. Чтобы констатировать неуспешность цели A, X, программа должна испробовать все деревья (5_j).

Предложение 3 можно схематически записать в виде

$$\frac{(\sim 1)Y;\quad (\sim 2)Y;\ \ldots}{(\sim)Y}\ (\sim_1).$$

Предложение 4. $(\sim j)A$, X *эквивалентно выполнению одного из следующих условий:*

(а) *имеется меньше чем* j *подходящих предложений или* j*-е подходящее предложение имеет вид* j: $A' \leftarrow Z$, *но* A *и* A' *не унифицируемы;*

(б) j*-е подходящее предложение имеет вид* j: $A' \leftarrow Z$, $s = MGU(A', A)$ *и верно* $(\sim)(ZX)s$.

Доказательство. При обработке дерева (5_j) сначала проверяется условие (а), и если оно не выполнено, то строится цель $(ZX)s$ в обозначениях условия (б). □

Предложение 4 можно оформить в виде двух правил.

$$\frac{\text{имеется } <i \text{ подходящих правил} \quad \text{или} \quad i: A' \leftarrow Z \text{ и } A', A \text{ не унифицируемы}}{(\sim i)A, X}\ (\sim_2)$$

$$\frac{i: A' \leftarrow Z;\quad (\sim)(ZX)s}{(\sim i)A, X}\ (\sim_3).$$

Теорема о полноте. *Цель* X *успешна тогда и только тогда, когда список* X *выводим по правилам* (RES), $(\sim_1)$, $(\sim_2)$, $(\sim_3)$ *и*

$$\frac{X+s}{X}\ (\text{fin}).$$

Доказательство получается из определения успешности цели и предложений 2—4. □

6. Теорема о полноте для языка с предикатом /

Расширим теперь язык, допуская атом / в правых частях предложений. При этом вместо X, /, Y будем писать просто X/Y.

Предикат / задает ограничение перебора в случае неудачи, т. е. отмены подстановки. Говоря неформально, отмена цели / приводит к отмене не только (подстановки, найденной для) ее левого брата, но и к неудаче той вершины, развертывание которой непосредственно вызвало появление этого /.

Точнее, мы модифицируем описание работы ПРОЛОГ-программы следующим образом.

Атом / всегда считается успешным, т. е. активная атомарная вершина / при работе алгоритма NEXT всегда помечается знаком +.

В пункте 2.2.1 описания алгоритма NEXT добавляем условие $B_k \neq /$ и добавляем новый пункт.

2.2.3. B_k есть /. Отменяем в рассматриваемом дереве (7) раздела 4 подстановки s, s_1, ..., s_k, удаляем всю часть, расположенную ниже вершины A, и заменяем A на $\sim A$.

Рассмотрим, как изменяется каждое из предложений 1—4.

Предложение 1 сохраняется и в присутствии предиката /. Действительно, если дерево (2) в разделе 5 успешно, то его обработка ПРОЛОГ-программой происходит так, как будто / просто не входит в

цель $A_kX_k \ldots X_1X_0$: ведь при развертывании дерева / всегда считается успешным (соответствующая ветвь немедленно закрывается), и ни одно вхождение / в цель $A_kX_k \ldots X_1X_0$ не отменяется, так как это означало бы неуспешность дерева (2) — отмену одного из правил $i_0, \ldots, i_k$ или всего дерева.

Прежде, чем рассмотреть аналоги предложений 2—4 из раздела 5, изменим понимание записи $(\sim i)Y$ и введем новый символ $(\sim i/)Y$.

$(\sim i)Y$ означает теперь, что дерево (5_i) из раздела 5 отменямое, причем отмена происходит не из-за /.

$(\sim i/)Y$ означает, что это дерево — отменяемое, и отмена происходит из-за /, находящегося в i-м предложении.

Теперь предложение 2 (раздел 5) сохраняется вместе с доказательством (для целей, не содержащих /), но приобретает несколько иной смысл: посылка $[\sim i]A, X$ в правиле (RES) означает, что первые i подходящих для A предложений данной программы приводят к неудаче, но не из-за /.

Предложение 3′. $(\sim)A, X$ *эквивалентно выполнению одного из следующих условий (где* k *обозначает количество подходящих для* A *предложений).*

(а) $[\sim k]A, X$;

(б) $[\sim i]A, X$ и $(\sim(i+1)/)A, X$ для некоторого $i<k$.

Доказательство. Чтобы констатировать неуспешность цели A, X программа перебирает деревья (5_i) и убеждается, что каждое из них отменяемо. Перебор заканчивается либо на последнем дереве (случай (а)), либо (случай (б)), если отмена происходит за счет /. □

Схематически предложение 3 формулируется как совокупность правила $(\sim_1)$ из раздела 5 (случай (а)) и правила

$$\frac{[\sim i]Y; \quad (\sim(i+1)/)Y}{(\sim)Y} \quad (\sim_4)$$

Предложение 4′. $(\sim j)A, X$ *эквивалентно выполнению одного из следующих условий:*

(а) *из предложения* 4 (раздел 5).

(б) *из предложения* 4 (раздел 5), *причем* Z *не содержит предиката* /.

(в) j*-е подходящее предложение имеет вид* j: $A' \leftarrow X' \,/\, X''$, *где* X' *не содержит* /, $s=MGU(A', A)$ *и верно* $(\sim)X's$.

Доказательство. При обработке дерева (5_j) сначала проверяется условие (а), и если оно не выполнено, то строится цель $(ZX)s$, где Z — правая часть j-го подходящего предложения (т. е. Z в случае (б) и $X' \,/\, X''$ в случае (в)). Затем начинается перебор целей Zs. Если Z не содержит /, то любая отмена дерева (5_j) дает $(\sim j)A, X$. Если же Z содержит / (случай (в)), то $(\sim j)A, X$ получается только из-за отмены при обработке целей $X's$, т. к. после перехода через / к целям $(X''X)s$ отмена вызовет $(\sim j/)A, X$.

Схематически предложение 4 представляется правилами $(\sim_2)$, $(\sim_3)$ из раздела 5 (последнее с ограничением: Z не содержит /) и правилом

$$\frac{j: A' \leftarrow X'/X''; \quad (\sim)X's; \quad s=MGU(A', A); \quad X' \text{ не содержит } /}{(\sim j)A, X} \quad (\sim_5)$$

Введение новых объектов $(i/)Y$ требует дополнительного критерия их истинности.

Предложение 5. $(\sim i/)A, X$ *эквивалентно выполнению условий:*

$$i: A' \leftarrow X'/X''; \quad X's+s'; \quad (\sim)(X'', X)ss', \qquad (1)$$

где $s=MGU(A', A)$.

Доказательство. Обработка дерева (5_i) раздела 5 начинается с замены A на X'/X'' и проведения подстановки $s=MGU(A', A)$ в X'/X'' и во всех оставшихся целях X. Затем начинается обработка целей $X's$. Она заканчивается успешно — в противном случае получилось бы $(\sim i)A, X$, а не $(\sim i/)A, X$. Обозначив через s' результирующую подстановку, мы видим, что дальше обрабатываются цели $(X'', X)ss'$. Этот этап заканчивается отменой — иначе дерево в целом было бы успешным. Но отмена после / как раз и дает $(\sim i/)A, X$.

Схематически:

$$\frac{(1)}{(\sim i/)A, X}\ (\sim_6).$$

Теорема о полноте при наличии /. *Цель X успешна тогда и только тогда, когда список X выводим из аксиомы $/\leftarrow$ по правилам* (RES), (fin), $(\sim_1)$ — $(\sim_6)$.

Доказательство получается из предложений 2, 3′, 4′, 5. □

7. Приложение. Описание исчисления

Термы строятся из переменных с помощью функциональных символов. *Атомы* имеют вид $P(\mathbf{t})$, где P — предикат, $\mathbf{t}$ — список термов. Имеется выделенный 0-местный предикат /. *Предложения* имеют вид

$$B \leftarrow Z, \qquad (1)$$

где Z — список атомов (быть может, пустой), B — атом или пустое слово. *Программа* — упорядоченный список предложений. Предложение (1) подходящее для атома A, если B — атом, начинающийся с того же предиката, что и A.

Предполагается, что зафиксирована некоторая программа P, в которой все предложения, начинающиеся одним и тем же предикатом, помещены рядом. Xs означает результат выполнения подстановки s в выражении X.

Выводимые объекты имеют вид

$$X, \quad X+s, \quad (\sim)X, \quad (\sim i)X, \quad (\sim i/)X,$$

где X — список атомов, s — подстановка, i — натуральное число.

$[\sim i]X$ — сокращение для $\underset{j\leqslant i}{\&}\ (\sim j)X$.

Аксиома. $/\leftarrow$

Правила вывода.

Обозначения. $i: A' \leftarrow Z$ есть i-е по порядку подходящее предложение для атома A; $s=MGU(A', A)$ — наиболее общий унификатор атомов A' и A.

$$\frac{(i+1): A' \leftarrow Z; \quad (ZX)s+s'; \quad [\sim i]A, X}{A, X+ss'}\ (\mathrm{RES})$$

$$\frac{X+s}{X}\ (\mathrm{fin})$$

$$\frac{(\sim 1)Y; \quad (\sim 2)Y; \ \ldots}{(\sim)Y}\ (\sim_1)$$

$$\frac{\text{имеется подходящих для } A \text{ правил } \begin{matrix} <i \\ i:A'\leftarrow Z \end{matrix} \qquad \text{или } A',\ A \text{ не унифицируемы}}{(\sim i)A, X} \ (\sim_2)$$

$$\frac{i:A'\leftarrow Z; \quad (\sim)(Z,X)s; \quad Z \text{ не содержит } /}{(\sim i)A, X} \ (\sim_3)$$

$$\frac{[\sim i]A, X; \quad (\sim (i+1)/)A, X}{(\sim)A, X} \ (\sim_4)$$

$$\frac{i:A'\leftarrow X'/X''; \quad (\sim)X's; \quad X' \text{ не содержит } /}{(\sim i)A, X} \ (\sim_5)$$

$$\frac{i:A'\leftarrow X'/X''; \quad X's+s'; \quad (\sim)(X'', X)ss'}{(\sim i/)A, X} \ (\sim_6)$$

ЛИТЕРАТУРА

1. *Clocksinn, W. F., Mellish, C. S.* Programming in PROLOG. Springer, 1981.
2. *Kowalski, R.* Logic for Problem Solving. North-Holland, 1979.
(Готовится перевод в изд-ве «Мир».)
3. *Apt, K. R., van Emden, M. H.* Communic. ACM, **29**, 841—862 (1982).

Институт кибернетики
Академии наук Эстонской ССР

Поступила в редакцию
31/X 1985

G. MINTS

PROLOGi TEATUD ALAMHULGAGA EKVIVALENTNE ARVUTUS

On kirjeldatud PROLOGi teatud alamhulgaga ekvivalentset arvutust.

G. MINTS

COMPLETE CALCULUS FOR PURE PROLOG

The calculus in question is tuned to a PROLOG-program P. Derivability of an atomic formula A in the calculus corresponding to P is equivalent to the success of the goal A for P.

Pure PROLOG means a subset without negation-as-failure operation NOT for atoms, executable predicates or predicates of predicates. The search strategy of PROLOG is known to be logically incomplete; on the other hand it includes certain semidecision procedure allowing to diagnose failure in some cases and to use this for further search. To take account of this feature as well as of the control operation /, a new kind of derivable objects $(\sim)\ X$ (negation-as-failure of the negative Horn clause X) is introduced and corresponding inference rules are written down.