ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆԻ ԳԻՏԱԿԱՆ ՏԵՂԵԿԱԳԻՐ
УЧЕНЫЕ ЗАПИСКИ ЕРЕВАНСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

Բնական գիտություններ 3, 2001 Естественные науки

Математика

УДК 519.682.1

А.М. АМБАРЦУМЯН

ОБ ИНТЕРПРЕТАТОРЕ ПРОСТОГО МОНАДИЧЕСКОГО ПРОЛОГА

В работе рассмотрены вопросы преобразования ПРОЛОГ-программ и ПРОЛОГ-запросов с целью повышения интеллектуальных возможностей интерпретатора. Для некоторого подмножества ПРОЛОГа доказаны невозможность преобразования программ и запросов к виду, позволяющему интерпретатору остановиться с определенным ответом, и возможность преобразования к виду, допускающему остановку в случае положительного ответа.

Введение. В данной работе обсуждаются вопросы, относящиеся к расширению интеллектуальных способностей логических систем программирования посредством преобразований их программ и запросов. Предметом нашего исследования является система ПРОЛОГ [1].

Известно (см. [2, 3]), что интерпретатор ПРОЛОГа является логически корректным (т. е. не лжет), однако не является логически полным (т. е. отвечает не на все запросы, являющиеся логическим следствием программы). С другой стороны, результат работы интерпретатора зависит от порядка предложений программы и от порядка атомов в телах предложений и запроса. В [4] было показано, что, переставляя и удаляя предложения ПРОЛОГ-программ и переставляя атомы в телах правил ПРОЛОГ-программ и ПРОЛОГ-запросов, интерпретатор ПРОЛОГа невозможно довести до логически полного уже в случае простого монадического ПРОЛОГа (т. е. ПРОЛОГа, не использующего встроенные предикаты и использующего только 0-местные функциональные символы и предикатные символы местности 0 и 1). Мы рассматриваем простой монадический ПРОЛОГ, программы которого используют не более одного повторяющегося предикатного символа в головах предложений. Доказывается, что, используя описанные выше преобразования, невозможно довести интерпретатор описанной версии ПРОЛОГа до разрешающего, но можно довести его до логически полного.

Работа состоит из введения и двух разделов. В первом разделе приведены используемые определения и результаты, во втором – изложены полученные результаты.

1.**Используемые понятия и результаты.** Зададимся тремя непересекающимися счетными множествами X, F_0, $П_1$. X – множество предметных переменных, F_0 – множество 0-местных функциональных символов, $П_1$ – множество 0- и 1-местных предикатных символов, такое, что подмножество предикатных символов каждой местности счетно. Дадим определение атома.

1. Всякий 0-местный предикатный символ из Π_1 есть атом.
2. Если p – 1-местный предикатный символ, а $t \in X \cup F_0$, то p(t) есть атом.
3. Других атомов нет.

Из атомов с помощью логических операций $\neg, \vee, \&, \supset$ и кванторов $\forall, \exists$ строятся формулы:

1. Каждый атом – формула.
2. Если A, B – формулы, x – переменная, то $\neg$ (A) , (A) & (B), (A) $\vee$ (B), (A) $\supset$ (B), $\forall x$ (A), $\exists x$ (B) – формулы.
3. Других формул нет.

Множество формул обозначим через Ф.

Опишем рассматриваемые нами интерпретации. Предметным множеством этих интерпретаций будет множество $M = F_0$. Каждому символу из F_0 сопоставляется он сам. Каждому 0-местному предикатному символу из Π_1 сопоставляется один из элементов множества {true, false}, а каждому 1-местному символу из Π_1 – некоторое отображение $M \to$ {true, false}. Обозначим описанное множество интерпретаций через H.

Пусть A, B $\in$ Ф. Будем говорить, что формула B является логическим следствием формулы A, и обозначать $A \models B$, если формула $(A) \supset (B)$ принимает значение *true* на всех интерпретациях из H.

Пусть $\Phi' \subset \Phi$, и $\Phi' \neq \varnothing$. Под преобразованием формул множества Φ' будем понимать пару (A, A'), где $A, A' \in \Phi'$ (см. [4]). Пусть T – некоторое непустое множество преобразований формул множества Φ'. Будем говорить, что формула $A \in \Phi'$ T-преобразуема в формулу $B \in \Phi'$ $(A \xrightarrow{T} B)$, если существует такая последовательность преобразований $(A_1, A_2), (A_2, A_3), \ldots, (A_{n-1}, A_n)$, принадлежащих T, что $A_1 = A$, $A_n = B$, $n > 1$. В этом случае формулу B назовем T-образом формулы A.

Определим множество программ *Prog* простого монадического ПРОЛОГа. Программа $P \in$ *Prog* отождествляется с формулой вида

$$\forall x_1, \ldots, \forall x_s\ (S_1 \,\&\, \ldots \,\&\, S_n),$$

где S_i – дизъюнкт Хорна $A_i \vee \neg B_{i1} \vee \ldots \vee \neg B_{im_i}$, A_i, B_{ij} – атомы, $j = 1, \ldots, m_i$, $m_i \geq 0$, $i = 1, \ldots, n$, $n > 0$, а $x_1, \ldots, x_r$ $(r \geq 0)$ – все переменные, использованные в дизъюнктах $S_1, \ldots, S_n$. В соответствии с синтаксисом ПРОЛОГа программа P записывается в виде последовательности дизъюнктов $S_1, \ldots, S_n$. Дизъюнкты называются предложениями и записываются в виде A_i :- $B_{i1}, \ldots, B_{im_i}$. Атом A_i называется головой, а последовательность $B_{i1}, \ldots, B_{im_i}$ – телом предложения S_i, $m_i \geq 0$, $i = 1, \ldots, n$. При $m_i = 0$ предложение S_i называется фактом, а при $m_i > 0$ – правилом.

Определим множество запросов *Quer* простого монадического ПРОЛОГа. Запрос $Q \in$ *Quer* отождествляется с формулой вида

$$\exists y_1, \ldots, \exists y_s\ (C_1 \,\&\, \ldots \,\&\, C_k),$$

где $C_1, \ldots, C_k$ – атомы, а $y_1, \ldots, y_s$ $(s \geq 0)$ – все переменные, использованные в них. В соответствии с синтаксисом ПРОЛОГа запрос Q записывается в виде ?-$C_1, \ldots, C_k$. Число k называют длиной запроса Q.

Опишем работу интерпретатора простого монадического ПРОЛОГа U. Для этого введем некоторые понятия.

Атомы A и B назовем унифицируемыми, а подстановку σ – их наиболее общим унификатором, если $A = p(t_1)$, $B = p(t_2)$ и

а) либо t_1 и t_2 – различные переменные и $\sigma = \{t_1/t_2\}$ (или $\sigma = \{t_2/t_1\}$);

б) либо t_1 – переменная, t_2 – 0-местный функциональный символ и $\sigma = \{t_2/t1\}$;

в) либо t_1 – 0-местный функциональный символ, t_2 – переменная и $\sigma = \{t_1/t_2\}$;

г) либо t_1 и t_2 – один и тот же 0-местный функциональный символ или одна и та же переменная, а $\sigma = \varepsilon$, где ε – пустая подстановка.

Введем понятие правила вывода ρ. Правило вывода ρ применимо к запросу Q: $?\text{-}C_1, \ldots, C_k$ $(k > 0)$ и предложению S: $A :\text{-} B_1, \ldots, B_m$ $(m \geq 0)$, если Q и S не имеют общих переменных и C_{i_0} с A унифицируемы (индекс i_0 определяется конкретным правилом ρ). В этом случае результат применения правила вывода есть запрос $?\text{-}\ C_1\sigma, \ldots, C_{i_0-1}\sigma, B_1\sigma, \ldots, B_m\sigma, C_{i_0+1}\sigma, \ldots, C_k\sigma$, который обозначается через $\rho(S, Q)$, где σ – наиболее общий унификатор C_{i_0} и A. Если в последовательности $Q_1, \ldots, Q_n$ запрос $Q_i+1 = \rho(S_{i_j}, Q_i)$, $1 \leq i < n$, где S_{i_j} – некоторое предложение программы P, то такую последовательность запросов называют ρ-выводом запроса Q_n из программы P и запроса Q_1 (или просто выводом, если нам не важно, какое именно правило ρ было использовано).

Работа интерпретатора ПРОЛОГа основана на правиле вывода ρ_1, которое всегда выбирает первый атом запроса, и на стратегии его применения. Стратегия применения правила вывода в случае интерпретатора ПРОЛОГа основана на порядке предложений программы. Функционирование интерпретатора заключается в конструировании то удлиняющейся, то укорачивающейся последовательности запросов. Если в последовательности появляется пустой запрос, то интерпретатор останавливается с положительным ответом, если же пустой запрос не появился и дальнейшее конструирование последовательности невозможно, то интерпретатор останавливается с отрицательным ответом. Также возможно и бесконечное функционирование интерпретатора. Опишем алгоритм работы интерпретатора простого монадического ПРОЛОГа (который получается из более общего алгоритма интерпретатора ПРОЛОГа, описанного в [3]). Пусть $P \in Prog$, $Q \in Quer$.

Шаг 1. Q_1 взять равным Q, $i := 1$, опустошить стек.

Шаг 2. Если Q_i – пустой запрос, то остановиться с положительным ответом, иначе перейти к шагу 3.

Шаг 3. Просмотреть предложения P согласно их порядку следования до первого такого предложения S_k, что правило вывода ρ_1 применимо к S_k и Q_i. Если такое S_k существует, то $Q_{i+1} := \rho_1(S_k, Q_i)$, $i := i+1$, k поместить в стек и перейти к шагу 2, иначе перейти к шагу 4.

Шаг 4. Если $i = 1$, то остановиться с отрицательным ответом, иначе перейти к шагу 5.

Шаг 5. Считать верхний элемент стека. Пусть он равен d. Просмотреть предложения P, начиная с S_d, согласно их порядку, до первого такого S_j $(j >$

d), что ρ_l применимо к S_j и Q_{i-1}. Если такое S_j существует, то заменить старый запрос Q_i на новый $Q_i = \rho_l(S_j, Q_{i-1})$, j поместить в стек и перейти к шагу 2, иначе ликвидировать Q_i, $i := i-1$ и перейти к шагу 4.

Результат применения U к P и Q обозначим $U(P, Q)$.

Воспользуемся также следующими понятиями и обозначениями, введенными в [4].

Пусть $P \in Prog$. Тогда

$$Yes(P) = \{Q \in Quer \mid P \models Q\},$$

$$Ans(U, P) = \{ Q \in Quer \mid U(P, Q) \text{ определено}\}.$$

Пусть P_1, $P_2 \in Prog$. Будем говорить, что P_1 и P_2 Δ-эквивалентны (обозначим $P_1 \overset{\Delta}{\sim} P_2$), если $Yes(P_1) = Yes(P_2)$.

Будем говорить, что запрос $?\text{-}C_1', \ldots, C_k'$ подобен запросу $?\text{-}C_1, \ldots, C_k$, если $C_1', \ldots, C_k'$ есть перестановка $C_1, \ldots, C_k$, $k > 0$. Будем говорить, что предложение A :- $B_1', \ldots, B_m'$ подобно предложению A :- $B_1, \ldots, B_m$, если $B_1', \ldots, B_m'$ есть перестановка $B_1, \ldots, B_m$, $m \geq 0$. Программу $P' = S_1', \ldots, S_n'$ назовем подобной программе $P = S_1, \ldots, S_n$, если предложение S_i' подобно предложению S_i, $i = 1, \ldots, n$, $n > 0$. Программу P' назовем перестановкой (подпоследовательностью) программы P, если последовательность предложений P' есть перестановка (подпоследовательность) последовательности предложений P.

Мы будем использовать следующую систему преобразований (см. [4]) программ $Prog$ и запросов $Quer$:

$T_1 = \{(P, P') \mid P, P' \in Prog, P' -$ перестановка $P\}$;
$T_2 = \{(P, P') \mid P, P' \in Prog, P'$ подобна $P\}$;
$T_3 = \{(P, P') \mid P, P' \in Prog, P' -$ подпоследовательностью $P\}$;
$T = T_1 \cup T_2 \cup T_3$;
$T_q = \{(Q, Q') \mid Q, Q' \in Quer, Q'$ подобен $Q\}$.

В [4] было доказано, что не для каждой программы $P \in Prog$ и не для каждого запроса $Q \in Quer$ существуют программа $P' \in Prog$ и запрос $Q' \in Quer$ такие, что $P \xrightarrow{T} P'$, $Q \; Q'$, $P \overset{\Delta}{\sim} P'$, $Q \in Yes(P) \Rightarrow Q' \in Ans(U, P')$. Иначе говоря, используя преобразования систем T и T_q, невозможно довести интерпретатор ПРОЛОГа до логически полного.

2. **Простой монадический пролог с одним повторяющимся предикатным символом.** Предикатный символ p, используемый в программе P, назовем повторяющимся, если число предложений P, головы которых используют p, больше 1. Обозначим через $Prog_1$ множество программ из $Prog$, использующих не более 1 повторяющегося предикатного символа.

Теорема 1. Не для каждой программы $P \in Prog_1$ и не для каждого запроса $Q \in Quer$ существуют программа P' и запрос Q' такие, что $P \xrightarrow{T} P'$, $Q \xrightarrow{T''} Q'$, $P \overset{\Delta}{\sim} P'$ и $Q' \in Ans(U,P')$.

Доказательство. Рассмотрим следующие программу P и запрос Q.

Программа P:

$p(a).$
$s(b) :- p(x).$
$u(a) :- p(x).$
$p(x) :- p(y).$

Запрос Q:

$?-s(x), u(x).$

Очевидно, что любой T-образ программы P есть подпоследовательность некоторой перестановки P, а любой T_q-образ запроса Q подобен Q. Легко видеть, что для любой перестановки P' программы P и для любого T_q-образа Q' запроса Q $Q' \notin Ans(U, P')$. Также легко заметить, что для любой собственной подпоследовательности P'' любой перестановки программы P $P \overset{\Delta}{\not\to} P''$.

Теорема 1 доказана.

Содержательный смысл теоремы 1 состоит в следующем: невозможно, используя преобразования систем T и T_q, довести интерпретатор простого монадического ПРОЛОГа до разрешающего в случае программ из $Prog_1$.

Теорема 2. Для любой программы $P \in Prog_1$ существует Δ-эквивалентный ей T-образ P' такой, что для любого запроса $Q \in Quer$ существует T_q-образ Q' такой, что $Q \in Yes(P) \Rightarrow Q' \in Ans(U, P')$.

Содержательно теорема 2 утверждает следующее: используя преобразования систем T и T_q, возможно довести интерпретатор простого монадического ПРОЛОГа до логически полного в случае программ из $Prog_1$.

Перед тем, как перейти к непосредственному доказательству теоремы 2, сформулируем ряд понятий.

Пусть $Q_1, \ldots, Q_n$ – вывод из программы P и запроса $?-u(x)$, и пусть Q_n использует переменную y. Будем говорить, что переменная y является потомком переменной x, если $\{ y / x \} \subset \sigma_1 \ldots \sigma_{n-1}$, где σ_i – подстановка, использованная для получения Q_{i+1} из Q_i, $i = 1, \ldots, n-1$. Также будем предполагать, что атомы в телах предложений и запросах не повторяются; очевидно, это предположение не влияет на суть проблемы.

Доказательство теоремы 2. Пусть $P \in Prog_1$ и p – повторяющийся предикатный символ программы P (если таковой имеется). Вначале покажем, что существует такой T-образ P' программы P, что для любого запроса Q, принадлежащего простому монадическому ПРОЛОГу, существует такой его T_q-образ Q', что $Q' \in Yes(P') \Rightarrow Q' \in Ans(U, P')$. После этого докажем Δ-эквивалентность исходной и полученной программ. Введем следующие понятия.

Предложения, использующие предикатный символ p в головной части, для краткости будем называть p-предложениями.

Вывод $Q_1, \ldots, Q_n$ из программы P и запроса Q_1 назовем атомарным, если $Q_i = ?-C_{i1}, \ldots C_{ik}$ и существует j, $1 \le j \le k$, $Q_{i+1} = \rho(S, ?- C_{ij})$ для некоторого предложения $S \in P$, $1 \le i \le n$.

Правило $A :- B_1, \ldots, B_m$ назовем бесполезным правилом первого рода, ес-

ли существует атомарный вывод $Q_1,..., Q_n$ из программы P и запроса $?\text{-}B_1,..., B_m$, не использующий p-предложений, но использующий предложение, в голове которого использован предикатный символ, входящий в состав запроса Q_n.

Правило A :- $B_1,..., B_m$, не являющееся бесполезным правилом первого рода, назовем бесполезным правилом второго рода, если A имеет вид $p(x)$, где p – повторяющийся предикатный символ, а x – переменная, и существует вывод $Q_1,..., Q_n$ из программы P и запроса $?\text{-}B_1,..., B_m$ такой, что Q_n использует $p(y)$, где y – потомок x.

Правило A :- $B_1,..., B_m$ назовем полезным правилом первого рода, если A не использует повторяющийся предикатный символ и A :- $B_1,..., B_m$ не является бесполезным правилом первого рода.

Правило A :- $B_1,..., B_m$ назовем полезным правилом второго рода, если A использует повторяющийся предикатный символ, A :- $B_1,..., B_m$ не является бесполезным правилом первого рода и никакой вывод из программы P и запроса $?\text{-}B_1,..., B_m$ не использует p-предложения.

Остальные правила назовем полезными правилами третьего рода. Легко видеть, что это такие правила A :- $B_1,..., B_m$, где A использует повторяющийся предикатный символ, и существует вывод $Q_1,..., Q_n$ из программы P и запроса $?\text{-}B_1,..., B_m$ такой, что Q_n использует p, но A :- $B_1,..., B_m$ не является бесполезным правилом второго рода.

Опишем преобразование программы P. Оно выполняется в два этапа. На первом этапе над каждым предложением S_i программы P выполняется следующее преобразование класса T_2. Для каждого атома вида $u(x)$ в теле S_i, где x – переменная, выполняется следующее: если существует вывод $Q_1,..., Q_n$ из программы P и запроса $?\text{-}u(x)$, не использующий p-предложений, такой, что Q_n использует атом $p(y)$, где y – либо потомок переменной x, либо сама переменная x, и ни один из остальных атомов в Q_n не использует y, то $u(x)$ переносится в конец тела предложения S_i. После этого, если среди оставшихся атомов имеется атом B, использующий головную переменную предложения (при условии, что в голове использована переменная, иначе этот шаг пропускается), и существует вывод из запроса $?\text{-}B$, не использующий p-предложений, в ходе которого переменная атома B получает значение, то такой атом переносится в начало тела предложения. Если таких атомов несколько, переносится один из них (любой). На этом преобразование T_2 завершается. Обозначим программу, полученную в результате этого преобразования, через P_1. Опишем второй этап преобразования, заключающийся в перестановке и удалении предложений программы P_1 (преобразования классов T_1 и T_3). Преобразованная программа P' начинается предложениями группы *Facts*, состоящей из всех фактов программы P_1. Взаимное расположение предложений в группе *Facts* произвольно. За группой *Facts* следует группа $Rules_1$, состоящая из всех полезных правил первого рода. Взаимное расположение предложений в $Rules_1$ произвольно. За $Rules_1$ следует группа $Rules_2$, состоящая из всех полезных правил второго рода. Взаимное расположение предложений в $Rules_2$ произвольно. За $Rules_2$ следует группа $Rules_3$, состоящая из некоторых полезных правил третьего рода. Опишем состав и порядок предложений $Rules_3$.

Объединим полезные правила третьего рода в подгруппы $Rules_{3,1}$,..., $Rules_{3,m}$ и $Rules_3'$, где $Rules_{3,i} = \{A$:- $B_1,...,B_n \mid U(Facts \cup Rules_1 \cup Rules_2$

$\cup(\bigcup_{j=1}^{i-1} Rules_{3,j})$, *?-*$B_1$*, ...,* B_n*) = yes}*, $Rules_3' = Rules_3 \setminus (\bigcup_{j=1}^{i-1} Rules_{3,j})$. Группа $Rules_3$ начинается предложениями подгруппы $Rules_{3,1}$, и для каждого $i=1,\ldots, m-1$ за подгруппой $Rules_{3,i}$ следует подгруппа $Rules_{3,i+1}$. Порядок предложений в каждой подгруппе произволен. Предложения программы P_1, не вошедшие в описанные группы, удаляются.

Преобразование T_q над запросом к уже преобразованной программе выполняется по тому же алгоритму, что и преобразование T_2. При этом запрос рассматривается как тело предложения, голова которого не использует переменную.

Можно показать, что P' и Q' и есть искомая программа и искомый запрос.

Теорема 2 доказана.

Замечание. Вместо преобразований T в предыдущей теореме можно взять $T_1 \cup T_2$. Для этого достаточно после проведенного преобразования добавить в конец програмы удаленные предложения (бесполезные правила первого и второго родов и подгруппу $Rules_3'$).

Проиллюстрируем на примере доказательство теоремы 2. Рассмотрим следующую программу *P*:

```
p(x) :- p(c).
p(b) :- p(a).
s(b).
r(a) :- p(x), w(x).
p(a) :- r(a).
p(b) :- s(b).
q(x) :- t(x).
t(x) :- q(x).
p(x) :- p(x).
p(x) :- v(x).
v(x) :- p(y), s(y), p(x), s(x).
w(x) :- s(b).
p(x) :- q(x).
```

После преобразования тел предложений четвертое и одиннацатое предложения принимают следующий вид соответственно:

```
r(a) :- w(x), p(x).
v(x) :- s(x), s(y), p(x), p(y).
```

Остальные предложения остаются без изменений.

Теперь произведем разделение предложений на группы. Группа *Facts* состоит из одного предложения *s(b)*. Бесполезные правила первого рода:

```
q(x) :- t(x).
t(x) :- q(x).
p(x) :- q(x).
```

Бесполезные правила второго рода:

```
p(x) :- p(x).
p(x) :- v(x).
```

Полезные правила первого рода (группа $Rules_1$):

```
r(a) :- w(x), p(x).
v(x) :- s(x), s(y), p(x), p(y).
w(x) :- s(b).
```

Полезные правила второго рода (группа $Rules_2$):

$p(b) :- s(b).$

Подгруппа $Rules_{3,1}$:

p(a) :- r(a).

Подгруппа $Rules_{3,2}$:

p(b) :- p(a).

Подгруппа $Rules_3'$:

p(x) :- p(c).

Таким образом, преобразованная программа примет следующий вид:

$s(b).$
$r(a) :- w(x), p(x).$
$v(x) :- s(x), s(y), p(x), p(y).$
$w(x) :- s(b).$
$p(b) :- s(b).$
$p(a) :- r(a).$
$p(b) :- p(a).$

Стоит также проиллюстрировать важность переноса атома, использующего переменную головного атома, в начало предложения. Рассмотрим следующую программу:

$s(b).$
$p(a).$
$u(a) :- p(x).$
$p(x) :- u(y), s(x).$
p(x) :- p(y).

Данная программа не нуждается в вышеописанном преобразовании, за исключением перестановки атомов $u(y)$ и $s(x)$ в теле четвертого предложения. Если обратиться к ней с запросом $?\text{-}p(c)$, то интерпретатор не остановится с положительным ответом, хотя запрос является логическим следствием программы. Для исправления положения нужно поменять местами атомы $u(y)$ и $s(x)$ в четвертом предложении.

Кафедра алгоритмических языков

Поступила 05.10.2000

ЛИТЕРАТУРА

1. **Clocksin W. F., Mellish C. S.** Programming in Prolog. Berlin: Springer-Verlag, 1984.
2. **Lloyd J. W.** Foundations of Logic Programming. Berlin: Springer-Verlag, 1984.
3. **Нигиян С. А.** – Программирование, 1994, №2, с. 64–73.
4. **Нигиян С. А., Хачоян Л. О.** – Программирование, 1997, №6, с. 17–28.

Ա.Մ. ՀԱՄԲԱՐՁՈՒՄՅԱՆ

ՊԱՐԶ ՄՈՆԱԴԻԿ PROLOG-Ի ՄԵԿՆԱԲԱՆԻՉԻ ՄԱՍԻՆ

Ամփոփում

Աշխատանքում դիտարկվում են PROLOG ծրագրերի և PROLOG հարցումների ձևափոխությունների հարցերը՝ մեկնաբանիչի ինտելեկտուալ հնարավորությունները բարձրացնելու նպատակով: PROLOG-ի որոշ ենթաբազմության համար ապացուցվում են հետևյալ արդյունքները.

ա) անհնար է ձևափոխել այդ ենթաբազմության կամայական ծրագիր և հարցում այնպես, որպեսզի ինտերպրետատորը կանգ առնի հստակ պատասխանով;

բ) եթե հարցումը հանդիսանում է ծրագրի տրամաբանական հետևանք, ապա այդպիսի ձևափոխումը հնարավոր է:

A. M. HAMBARDZUMYAN

ON SIMPLE MONADIC PROLOG INTERPRETER

Summary

In this paper we discuss questions concerning transformations of PROLOG-programs and PROLOG-queries intended for extension of intellectual abilities of the interpreter. Some subset of simple monadic PROLOG is considered. It is proved that one cannot transform any program and query from this subset in such a way that the interpreter will always stop with a definite answer, but it is possible to do it if the query is a logical consequence of the program.