ՀԱՅԱՍՏԱՆԻ ԳԻՏՈՒԹՅՈՒՆՆԵՐԻ ԱԶԳԱՅԻՆ ԱԿԱԴԵՄԻԱՅԻ ԶԵԿՈՒՅՑՆԵՐ
ДОКЛАДЫ НАЦИОНАЛЬНОЙ АКАДЕМИИ НАУК АРМЕНИИ

Том 96 1996 №2-4

МАТЕМАТИКА

УДК 519.68:510

С. А. Нигиян, Л. О. Хачоян

Интерпретация и преобразование логических программ

(Представлено академиком НАН Армении Н.У.Аракеляном 30/V 1996)

Предлагаемая работа относится к математическим основам логического программирования. Логический язык программирования определяется как язык, программами которого являются выполнимые формулы некоторой логики. К программе обращаются с запросами. Интерпретатор логического языка программирования по программе и соответствующему ей запросу решает: является ли запрос логическим следствием программы? Естественно, что интерпретатор должен быть логически корректным, т.е. если он не в силах ответить на поставленный запрос, то должен либо остановиться с неопределенным ответом, либо функционировать бесконечно (см.([1])). Однако область определения интерпретатора можно расширять, осуществляя преобразования логических программ (см. ([2])). Естественно, преобразования должны быть такими, чтобы множества запросов, являющихся логическим следствием исходной и преобразованной программ, совпадали.

В разделе 1 данной статьи описан общий подход к обсуждаемой проблеме, в разделе 2 получены конкретные результаты для некоторых классов ПРОЛОГ-программ

1. Зафиксируем множество M, содержащее по крайней мере два элемента true и false. Введем понятие типа: 1) множество M есть тип; 2) любое непустое подмножество типа есть тип; 3) если $\beta,\alpha_1,\ldots,\alpha_k$ $(k>0)$ — типы, то множество всех отображений из $\alpha_1\times\ldots\times\alpha_k$ в β есть тип; 4) других типов, кроме определенных согласно 1-3, нет.

Пусть C — множество констант, принадлежащих некоторым типам, X — множество переменных некоторых типов. Из элементов множеств C и X строятся термы. Определение терма дано в работе ([1]). Элементарной формулой назовем терм, тип которого входит в множество {true, false}. Определение формулы: 1) любая элементарная формула есть формула, 2) если A, B — формулы и $x\in X$, то $\neg(A),(A)\&(B)$, $(A)\vee(B),(A)\supset(B),(A)\sim(B),\forall x(A),\exists x(A)$ есть формулы; 3) других формул, кроме определенных согласно 1-2, нет. Через $\Phi(C,X)$ обозначим

множество формул, построенных с использованием C и X. Некоторое вхождение переменной в формулу называется связанным, если оно находится в области действия λ или одного из кванторов $\forall$ или $\exists$. В противном случае это вхождение переменной называется свободным. Переменная x свободна в формуле A, если x имеет свободное вхождение в A. Интерпретация I формулы A определяется заданием значений всем ее свободным переменным. Значение формулы A на интерпретации I определяется естественным образом. Формула A называется выполнимой, если существует такая интерпретация I, на которой A принимает значение true. Формулу A будем называть тождественно истинной, если значение формулы A на любой интерпретации есть true. Если A, B – формулы и формула $(A)\supset(B)$ тождественно истинна, то будем говорить, что формула B является логическим следствием формулы A и обозначать $A \models B$.

Пусть $\Phi' \subset \Phi(C, X)$ и $\Phi' \neq \varnothing$. Под преобразованием формул множества Φ' мы будем понимать пару (A, A'), где $A, A' \in \Phi'$. Пусть T – некоторое непустое множество преобразований формул множества Φ'. Будем говорить, что формула $A \in \Phi'$ T-преобразуема в формулу $B \in \Phi'$ (обозначим $A \xrightarrow{T} B$), если существует такая последовательность преобразований $(A_1, A_2), \ldots, (A_{n-1}, A_n)$, принадлежащих T, что $A_1 = A$, $A_n = B$, $n > 1$.

Напомним определение логического языка программирования, взятое из [1]. Логический язык программирования определяется заданием шестерки $M, C, X, \mathrm{Prog}, \Delta, U$, где:

M – непустое множество, содержащее по крайней мере два элемента true и false;

C – множество констант, принадлежащих некоторым типам;

X – множество переменных некоторых типов;

Prog – множество программ, представляющее собой некоторое подмножество выполнимых формул из $\Phi(C, X)$;

Δ – отображение, которое каждой программе $P \in \mathrm{Prog}$ сопоставляет множество формул $\Delta(P) \subset \Phi(C, X)$, называемых запросами, соответствующими программе P;

U – интерпретатор, представляющий собой алгоритм, который для каждой программы $P \in \mathrm{Prog}$ и запроса $Q \in \Delta(P)$ либо останавливается с положительным ответом (yes), либо – с отрицательным (no), либо – с неопределенным ответом, либо функционирует бесконечно. Результат применения U к P и Q обозначим $U(P, Q)$. Если $U(P, Q)$ есть yes или no, то будем говорить, что $U(P, Q)$ определено. Интерпретатор U должен обладать свойством логической корректности:

если $U(P, Q) = \mathrm{yes}$, то $P \models Q$;

если $U(P, Q) = \mathrm{no}$, то $P \not\models Q$.

Пусть $P \in \mathrm{Prog}$. Введем обозначения:

$\mathrm{Yes}(P) = \{Q \in \Delta(P) \mid P \vDash Q\}$,

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

Интерпретатор U назовем *разрешающим* (соответственно *логически полным*), если для любой программы $P \in \mathrm{Prog}$ имеем: $\mathrm{Ans}(U, P) = \Delta(P)$ (соответственно $\mathrm{Yes}(P) \subset \mathrm{Ans}(U, P)$).

Введем отношение Δ-эквивалентности на множестве программ Prog. Пусть $P_1, P_2 \in \mathrm{Prog}$. Будем говорить, что P_1 и P_2 Δ-эквивалентны (обозначим $P_1 \overset{\Delta}{\sim} P_2$), если $\Delta(P_1) = \Delta(P_2)$ и $\mathrm{Yes}(P_1) = \mathrm{Yes}(P_2)$.

Пусть T – некоторое множество преобразований программ Prog, которое для каждой программы $P \in \mathrm{Prog}$ содержит преобразование (P, P).

Интерпретатор U назовем *T-разрешающим* (соответственно *T-логически полным*), если для любой программы $P \in \mathrm{Prog}$ существует такая программа $P' \in \mathrm{Prog}$, что $P \xrightarrow{T} P'$, $P \overset{\Delta}{\sim} P'$ и $\mathrm{Ans}(U, P') = \Delta(P')$ (соответственно $\mathrm{Yes}(P') \subset \mathrm{Ans}(U, P')$).

Пусть Quer есть множество, состоящее из запросов, соответствующих всем программам множества Prog. Пусть T_q – некоторое множество преобразований запросов Quer, которое для каждого запроса $Q \in \mathrm{Quer}$ содержит преобразование (Q, Q).

Интерпретатор U назовем *T, T_q-разрешающим* (соответственно *T, T_q-логически полным*), если для любой программы $P \in \mathrm{Prog}$ и любого запроса $Q \in \Delta(P)$ существуют такие $P' \in \mathrm{Prog}$ и $Q' \in \Delta(P')$, что $P \xrightarrow{T} P'$, $Q \xrightarrow{T_q} Q'$, $P \overset{\Delta}{\sim} P'$, $Q \in \mathrm{Yes}(P) \Leftrightarrow Q' \in \mathrm{Yes}(P')$ и $Q' \in \mathrm{Ans}(U, P')$ (соответственно $Q' \in \mathrm{Yes}(P') \Rightarrow Q' \in \mathrm{Ans}(U, P')$).

Легко видеть, что имеет место следующее:

$$
\begin{array}{lll}
U\text{-разрешающий} & \Rightarrow U\text{–}T\text{-разрешающий} & \Rightarrow U\text{–}T, T_q\text{-разрешающий} \\
\Downarrow & \Downarrow & \Downarrow \\
U\text{-логически полный} & \Rightarrow U\text{–}T\text{-логически полный} & \Rightarrow U\text{–}T, T_q\text{-логически полный}
\end{array}
$$

2. В данном разделе рассматривается ПРОЛОГ, не использующий предикаты от предикатов и встроенных предикатов – чистый ПРОЛОГ. Определим шестерку $M, C, X, \mathrm{Prog}, \Delta, U$, соответствующую чистому ПРОЛОГу.

Зададимся тремя непересекающимися счетными множествами F, Π и V. F – множество функциональных символов с приписанной каждому символу местностью, причем для любого $n \geq 0$ F содержит счетное число символов местности n. Π – множество предикатных символов с приписанной каждому символу местностью, причем для любого $n \geq 0$ Π содержит счетное число символов местности n. V – множество

предметных переменных. Из элементов множеств F и V строятся функциональные термы (ф.термы):

1) каждый 0-местный символ из F есть ф.терм;

2) каждая предметная переменная из V есть ф.терм;

3) если $n>0$, $t_1,\ldots,t_n$ – ф.термы и f – n-местный символ из F, то $f(t_1,\ldots,t_n)$ – ф.терм;

4) никаких других ф.термов, кроме определенных согласно 1-3, нет.

Через M_1 обозначим множество всех ф.термов, не использующих предметных переменных.

Предикатный терм (атом) определяется традиционным образом:

1) каждый 0-местный символ из П есть атом;

2) если $n>0$, $t_1,\ldots,t_n$ – ф.термы и p – n-местный символ из П, то $p(t_1,\ldots,t_n)$ – атом;

3) никаких других атомов, кроме определенных согласно 1-2, нет.

$M = M_1 \cup \{\text{true, false}\}$.

$C=F$, где каждый функциональный символ $f \in F$ местности $n>0$ есть отображение $M_1^n \to M_1$, которое n-ке $(t_1,\ldots,t_n)$, где $t_i \in M_1$, $i=1,\ldots,n$, ставит в соответствие ф.терм $f(t_1,\ldots,t_n)$.

$X = V \cup \Pi$, где тип каждой предметной переменной из V есть M_1, тип каждого 0-местного предикантного символа из П есть {true, false}, а тип каждого n-местного ($n>0$) предикатного символа из П есть множество всех отображений вида $M_1^n \to \{\text{true, false}\}$.

Определим множество программ Prog. Программа $P \in \text{Prog}$ есть последовательность предложений $S_1,\ldots,S_n$, $n>0$. Каждое предложение S_i входящее в P, имеет вид $A{:-}B_1,\ldots,B_m$ и представляет собой дизъюнкт $A \vee \neg B_1 \vee \ldots \vee \neg B_m$, где A, B_i – атомы, $i=1,\ldots,m$, $m \geq 0$. Атом A назовем головой, а последовательность $B_1,\ldots,B_m$ – телом предложения S. Число m назовем длиной тела предложения S. Если $m=0$, то S называется фактом, если же $m>0$, то S называется правилом. Пусть $\{x_1,\ldots,x_r\}$ – множество всех предметных переменных из V, входящих в программу P, $r \geq 0$. Программа P отождествляется с формулой:

$$\forall x_1 \ldots \forall x_r (S_1 \& \ldots \& S_n).$$

Опишем множество запросов Quer. В нашем случае $\Delta(P) = \text{Quer}$ для любой программы $P \in \text{Prog}$. Запрос $Q \in \text{Quer}$ имеет вид: $?{-}C_1,\ldots,C_k$, где C_i – атом, $i=1,\ldots,k$, $k>0$. Число k назовем длиной запроса Q. Пусть $\{y_1,\ldots,y_s\}$ – множество всех предметных переменных из V, входящих в запрос Q, $s \geq 0$. Запрос Q отождествляется с формулой:

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

U – является интерпретатором системы ПРОЛОГ, описание которого дано, например, в ([2]). Из ([2]) следует, что интерпретатор U в случае чистого ПРОЛОГа является логически корректным и не является логически полным. Отметим также, что результат интерпретатора U в случае чистого ПРОЛОГа зависит от порядка предложений программы, от порядка атомов в телах правил программы, от порядка атомов в запросе, от удаления некоторых предложений программы (естественно, сохраняющего Δ-эквивалентность программ).

Теорема 1. *Отношение Δ-эквивалентности и его дополнение не являются частично разрешимыми в случае чистого ПРОЛОГа.*

Перед тем как определить множества преобразований программ Prog и запросов Quer, введем ряд понятий.

Будем говорить, что запрос $?-C'_1,\ldots,C'_k$ подобен запросу $?-C_1,\ldots,C_k$, 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 программы P' есть перестановка (подпоследовательность) предложений программы P.

Определим следующие множества преобразований программ Prog:

$T_1=\{(P,P')|P,\ P'\in \text{Prog},\ P'$ есть перестановка $P\}$,

$T_2=\{(P,P')|P,\ P'\in \text{Prog},\ P'$ подобна $P\}$,

$T_3=\{(P,P')|P,\ P'\in \text{Prog},\ P'$ есть подпоследовательность $P\}$,

$T=T_1\cup T_2\cup T_3$.

Определим следующее множество преобразований запросов Quer:

$T_q=\{(Q,Q')|Q,\ Q'\in \text{Quer},\ Q'$ подобен $Q\}$

Чистый ПРОЛОГ, не использующий предметных переменных, назовем чистым ПРОЛОГом без переменных.

Теорема 2. *Интерпретатор U в случае чистого ПРОЛОГа без переменных является:*

T_1-логически полным,

не $T_1\cup T_2, T_q$-разрешающим,

не T_2, T_q-логически полным,

T_3-разрешающим.

Будем говорить, что программа P содержит повторяющийся предикатный символ p, если число предложений программы P, головы которых используют символ p, больше 1.

Чистый ПРОЛОГ, программы которого не содержат повторяющихся предикатных символов, назовем чистым ПРОЛОГом без повторяющихся предикатных символов.

Теорема 3. *Интерпретатор U в случае чистого ПРОЛОГа без повторяющихся предикатных символов является:*

логически полным,

не $T_1 \cup T_2, T_q$ -разрешающим,

T_3 -разрешающим.

Чистый ПРОЛОГ, использующий только 1-местные предикатные символы, 0-местные функциональные символы и предметные переменные, назовем монадическим ПРОЛОГом. Монадический ПРОЛОГ, длины запросов и тел правил которого равны 1, назовем обрезанным монадическим ПРОЛОГом.

Теорема 4. *Интерпретатор U не является T, T_q -логически полным в случае обрезанного монадического ПРОЛОГа, программы которого содержат не более двух повторяющихся предикатных символов*

Следствие теоремы 4. *Интерпретатор U не является T, T_q -логически полным в случае обрезанного монадического ПРОЛОГа.*

Теорема 5. *Интерпретатор U в случае обрезанного монадического ПРОЛОГа, программы которого содержат не более одного повторяющегося предикатного символа, является:*

T_1 -логически полным,

не T_1, T_q -разрешающим,

не T_3, T_q -логически полным,

$T_1 \cup T_3$ -разрешающим.

Ереванский государственный университет

Ս. Ա. ՆԻԳԻՅԱՆ, Լ. Օ. ԽԱՉՈՅԱՆ

Տրամաբանական ծրագրերի ինտերպրետացիա և ձևափոխություն

Աշխատանքում դիտարկվում են հարցեր, որոնք կապված են տրամաբանական ծրագրավորման լեզվի ինտերպրետատորի որոշման տիրույթի ընդլայնման հետ՝ տրամաբանական ծրագրերի ձևափոխման միջոցով: Բնականաբար, ձևափոխությունները պետք է լինեն այնպիսին, որ համընկնեն հարցումների այն բազմություններ, որոնք հանդիսանում են տրված և ձևափոխված ծրագրերի տրամաբանական հետևանքներ: Նկարագրված է դիտարկվող պրոբլեմի ընդհանուր մոտեցումը և ստացված են կոնկրետ արդյունքներ ՊՐՈԼՈԳ ծրագրերի որոշակի դասերի համար:

ЛИТЕРАТУРА – ԳՐԱԿԱՆՈՒԹՅՈՒՆ

[1] *С.А.Нигиян*, ДНАН Армении, т.95, №1, с.26-29 (1995). [2] *С.А.Нигиян*, Программирование, №2, 1994, с 64-73, (англ. перевод: *S.A.Nigiyan*, Programming and Computer Software, v.20, №2, p 69-75 (1994)).