ԵՐԵՎԱՆԻ ՊԵՏԱԿԱՆ ՀԱՄԱԼՍԱՐԱՆ

Նիգիյան Արամ Վիգենի

ԾՐԱԳՐԱՎՈՐՄԱՆ ՊՐՈՑԵԴՈՒՐԱՅԻՆ ԼԵԶՈՒՆԵՐԻ ԵՎ ՈՉ ԴԵՏԵՐՄԻՆԱՑՎԱԾ ՎԵՐՋԱՎՈՐ ԱՎՏՈՄԱՏՆԵՐԻ ՏՐԱՄԱԲԱՆԱԿԱՆ ՄՈԴԵԼԱՎՈՐՈՒՄ

Ա.01.09 «Մաթեմատիկական կիբեռնետիկա և մաթեմատիկական տրամաբանություն» մասնագիտությամբ
ֆիզիկամաթեմատիկական գիտությունների թեկնածուի
գիտական աստիճանի հայցման ատենախոսության

Ս Ե Ղ Մ Ա Գ Ի Ր

Երևան-2009

ЕРЕВАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Нигиян Арам Вигенович

ЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ ПРОЦЕДУРНЫХ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ И НЕДЕТЕРМИНИРОВАННЫХ КОНЕЧНЫХ АВТОМАТОВ

А В Т О Р Е Ф Е Р А Т

диссертации на соискание ученой степени кандидата
физико-математических наук
по специальности 01.01.09 “Математическая кибернетика и
математическая логика”

Ереван-2009

Ատենախոսության թեման հաստատվել է Երևանի պետական համալսարանում

Գիտական ղեկավար՝ ֆիզ.մաթ. գիտ. դոկտոր Ս.Ա.Նիգիյան

Պաշտոնական ընդդիմախոսներ՝ ֆիզ.մաթ. գիտ. դոկտոր Հ.Բ. Մարանջյան

ֆիզ.մաթ. գիտ. թեկնածու Ա.Շ. Մալխասյան

Առաջատար կազմակերպություն՝ ՀՀ ԳԱԱ Ինֆորմատիկայի և ավտոմատացման պրոբլեմների ինստիտուտ

Պաշտպանությունը կայանալու է 2009 թ. մարտի 4-ին ժամը 14^{00}-ին ԵՊՀ-ում գործող ԲՈՀ-ի 044 «Մաթեմատիկական կիբեռնետիկա և մաթեմատիկական տրամաբանություն» մասնագիտական խորհրդի նիստում, հետևյալ հասցեով՝ 0025, Երևան, Ալեք Մանուկյան 1:

Ատենախոսությանը կարելի է ծանոթանալ Երևանի պետական համալսարանի գրադարանում:

Սեղմագիրն առաքված է 2009 թ. փետրվարի 3-ին

Մասնագիտական խորհրդի գիտական քարտուղար՝
ֆիզ.մաթ. գիտ. թեկնածու Վ.Ժ. Դումանյան

Тема диссертации утверждена в Ереванском государственном университете

Научный руководитель: доктор физ.-мат. наук С. А. Нигиян

Официальные оппоненты: доктор физ.-мат. наук Г.Б. Маранджян

кандидат физ.-мат. наук А.Ш. Малхасян

Ведущая организация: Институт проблем информатики и автоматизации НАН РА

Защита диссертации состоится 4-го марта 2009 г. В 14^{00} часов на заседаниии специализированного совета ВАК 044 “Математическая кибернетика и математическая логика” при ЕГУ по адресу: 375025 г. Ереван, ул. Алека Манукяна, 1.

С диссертацией можно ознакомиться в библиотеке Ереванского государственного университета.

Автореферат разослан 3-го февраля 2009 г.

Ученый секретарь специализированного совета,
кандидат физ.-мат. наук В.Ж. Думанян

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность темы. Тема диссертационной работы посвящена логическому моделированию процедурных языков программирования и недетерминированных конечных автоматов. Процедурные программы представлялись в виде логических формул в работах Р. Флойда, С. Хоора, З. Манны для исследования логической семантики языков программирования. Представление недетерминированных конечных автоматов в виде логических программ можно найти в работе[1]. Представляет интерес определить процедурные языки программирования и недетерминированные конечные автоматы как логические языки программирования, что даст возможность исследовать свойства интерпреторов этих языков программирования такие как: непротиворечивость, полнота и др. Естестественно, что должна быть доказана корректность такого определения, т. е. в каком смысле логический язык программирования соответствует моделируемым нами объектам. Очевидно, что определяя логический язык программирования для рассматриваемых нами объектов, мы получаем возможность исследовать различные свойства моделируемых объектов, используя возможности соответствующего им логического языка программирования. Подобные исследования полезны еще и тем, что дают возможность осознанно включать рассматриваемые нами объекты в базы знаний, заранее зная, что "можно ожидать" от того или иного интерпретатора.

Цель диссертационной работы. Целью диссертационной работы является:

- Дать определение логического языка программирования, основанного на логике предикатов первого порядка. Как пример таких языков, рассмотреть языки хорновского программирования.
- Процедурный язык программирования определить как логический язык программирования, основанный на логике предикатов первого порядка. Доказать коррекность такого определения. Исследовать интерпретатор процедурного языка программирования с логической точки зрения.
- Недетерминированные конечные автоматы определить как язык хорновского программирования. Доказать корректность такого определения. Исследовать интерпретатор системы Пролог, как интерпретатор логического языка программирования, соответствующего недетерминированным конечным автоматам.

Методика исследований. Методика исследований включает в себя методы теории алгоритмов, математической логики, а также методы, используемые в теории автоматов и логическом программировании.

Научная новизна.

1. Дано определение логического языка программирования, основанного на логике предикатов первого порядка. Как пример логическогих языков программирования, основанных на логике предикатов первого порядка,

---

[1] *Bratko I.* Prolog Programming for Artificial Intelligence. // Addison-Wesley Pub. Comp., Inc., 1986 (рус. пер. *Братко И.* Программирование на языке Пролог для искусственного интеллекта. // М: Мир,1990, 560 с.).

рассматриваются языки хорновского программирования. Обсуждается язык хорновского программирования ПРОЛОГ и его интерпретатор U_0.

2. Процедурный язык программирования определяется как логический язык программирования, основанный на логике предикатов первого порядка. Доказывается коррекность такого определения. Доказывается также, что всякая процедурная программа имеет наименьшую модель и что интерпретатор процедурного языка программирования является логически полным и непротиворечивым.

3. Недетерминированные конечные автоматы определяются как язык хорновского программирования. Доказываются корректность такого определения и неполнота интерпретатора U_0 для данного языка. Доказывается также, что: существует недетерминированный конечный автомат, не имеющий оптимального для U_0 стандартного представления; каждый детерминированный конечный автомат имеет оптимальное для U_0 стандартное представление; всякое стандартное представление конечного автомата является оптимальным для U_0.

Теоретическая и практическая ценность. Результаты, полученные в диссертационной работе, носят как теоретический, так и практический характер. Они могут быть использованы при включении рассматриваемых объектов в базы знаний, а также при создании интерпретаторов логических систем программирования.

Апробация работы и публикации. Основные результаты диссертации докладывались на семинаре кафедры программирования и информационных технологий ЕГУ, на общем семинаре факультета информатики и прикладной математики ЕГУ, на годичных конференциях РАУ, 2006, 2008, на международной конференции "Conference on Computer Science and Information Technologies (CSIT'07)", Ереван, 2007.

Основные результаты работы отражены в 3-х публикациях.

Структура и объем работы. Диссертация состоит из введения, четырех глав, заключения и списка используемой литературы (35 наименований). Объем работы - 112 страниц.

СОДЕРЖАНИЕ РАБОТЫ

Во введении описывается задача диссертационного исследования обосновывается актуальность темы диссертации и её новизна. Дается краткий обзор содержания работы.

ГЛАВА 1. В данной главе дается определение логического языка программирования, основанного на логике предикатов первого порядка. Как пример логических языков программирования, основанных на логике предикатов первого порядка, рассматриваются языки хорновского программирования. Далее обсуждается язык хорновского программирования – ПРОЛОГ и его интерпретатор. Глава 1 состоит из трех разделов 1.1-1.3.

В разделе 1.1 дается определение логического языка программирования, основанного на логике предикатов первого порядка. Зафиксируем пять непересекающихся множеств M, V, $П$, $ВП$, $Ф$, где: M – непустое множество предметов, V – множество предметных переменных, $П$ – множество предикатных

символов, с приписанной каждому символу местностью, $B\Pi$ – множество встроенных предикатов, т.е. отображений вида $M^k \to \{true, false\}$, $k \geq 1$, Φ – множество встроенных функций, т.е. отображений вида $M^k \to M$, $k \geq 1$. Из элементов множеств M, V, Φ строятся термы. Атом определяется традиционным образом. Традиционным образом определяем формулу логики предикатов первого порядка.

Опишем рассматриваемые нами интерпретации. Предметным множеством рассматриваемых интерпретаций будет множество M. Каждому 0-местному символу из Π сопоставляется один из элементов множества $\{true, false\}$, а каждому k-местному $(k>0)$ символу из Π сопоставляется некоторое отображение: $M^k \to \{true, false\}$. Обозначим описанное множество интерпретаций через $Int(M)$. Отметим, что интерпретации из $Int(M)$ могут отличаться одна от другой лишь значениями, сопоставляемыми символам множества Π. Поэтому каждую интерпретацию $I \in Int(M)$ можно отождествить с подмножеством атомов вида q, где q – 0-местный символ из Π, и $p(m_1, \ldots, m_k)$, где p – k-местный $(k>0)$ символ из Π, $m_1, \ldots, m_k \in M$, значения которых на интерпретации I есть $true$. Легко видеть, что множество $Int(M)$ будет полной решёткой, если в качестве частичного порядка на $Int(M)$ взять отношение включения.

Пусть A замкнутая формула, и I интерпретация из $Int(M)$. Через $I(A)$ условимся обозначать значение формулы A на интерпретации I. Если $I(A)=true$, то интерпретация I называется моделью формулы A. Формула A называется выполнимой, если для неё существует модель. Пусть A и B замкнутые формулы. Будем говорить, что формула B логически следует из формулы A, и обозначать это $A \models B$, если любая интерпретация из $Int(M)$ является моделью формулы $A \supset B$.

Логический язык программирования, основанный на логике предикатов первого порядка определяется заданием семерки $M, V, \Pi, B\Pi, \Phi, Prog, \Delta$, первые пять компонент которой уже определены. Определим последние две: $Prog$ – множество программ, представляющее собой некоторое непустое множество выполнимых формул; Δ - отображение, которое каждой программе $P \in Prog$ сопоставляет непустое множество формул $\Delta(P)$, называемое множеством запросов, соответствующих программе P.

Интерпретатор логического языка программирования U для каждой программы $P \in Prog$ и запроса $Q \in \Delta(P)$ либо останавливается с положительным ответом, либо – с отрицательным ответом, либо функционирует бесконечно. Если U останавливается на P и Q, то результат применения U к P и Q обозначим $U(P,Q)$. Интерпретатор U должен быть логически непротиворечивым, т.е. обладать следующими свойствами: если U останавливается на P и Q с отрицательным ответом, т. е. $U(P,Q)=no$, то $P \models\!\!\not\models Q$; если U останавливается на P и Q с положительным ответом без значения, т.е. $U(P,Q)=yes$, то $P \models Q$; если U останавливается на P и Q с положительным ответом со значением, т.е. $U(P,Q)=\langle m_1,\ldots,m_k\rangle$, где $m_1,\ldots,m_k \in M$, $k \geq 1$, то запрос Q имеет вид $\exists x_1 \ldots \exists x_k A(x_1,\ldots,x_k)$ и $P \models A(m_1,\ldots,m_k)$. Интерпретатор U назовём логически полным, если из того, что $P \models Q$ следует, что U останавливается на P и Q с положительным ответом.

В разделе 1.2 как пример языков логического программирования, основанных на логике предикатов первого порядка, рассматриваются языки (чистого, без встроенных предикатов) хорновского программирования. Зафиксируем три непересекающихся множества V, Φ и Π, где V – множество предметных переменных, Π - множество предикатных символов с приписанной каждому символу местностью, Φ - множество функциональных символов с приписанной каждому символу местностью. Универсум Эрбрана M (см.[1]) определяется традиционным образом:

1. если a - 0-местный символ из Φ, то $a \in M$;

2. если $t_1,\ldots t_n \in M$ $(n \geq 1)$ и f - n-местный символ из Φ, то $f(t_1,\ldots, t_n) \in M$.

Считаем, что функциональные символы проинтерпретированы следующим образом: каждому 0-местному символу из Φ сопоставлен он сам, каждому n-местному $(n>0)$ символу $f \in \Phi$ сопоставлено отображение $M^n \to M$, которое n-ке $(t_1,\ldots,t_n) \in M^n$, где $t_i \in M$, $i=1,\ldots,n$, ставит в соответствие терм $f(t_1,\ldots,t_n)$.

Хорновская логическая программа P (далее просто программа) есть последовательность предложений $S_1,\ldots,S_n$, $n>0$. Предложение $S \in \{S_1,\ldots,S_n\}$ является либо фактом A, либо правилом $A\text{:-}B_1,\ldots,B_m$, которое является импликацией $B_1\&\ldots\&B_m \supset A$, где $A, B_1,\ldots,B_m$ – атомы, $m>0$. Можно считать, что фактом является предложение $A\text{:-}B_1,\ldots,B_m$, при $m=0$. Атом A называется головой предложения S, а последовательность $B_1,\ldots,B_m$ – его телом, очевидно, что факту соответствует пустое тело. Программе P сопоставляется формула:

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

где $x_1, \ldots, x_v$ – все переменные, использованные в предложениях $S_1, \ldots, S_n$, $v \geq 0$.

Хорновский запрос (далее просто запрос) Q имеет вид $?\text{-}C_1,\ldots,C_k$, где C_i – атом, $i=1,\ldots,k$, $k \geq 0$. Если $k=0$, то запрос называется пустым. Пустой запрос имеет вид $?\text{-}$. Непустому запросу Q сопоставляется формула:

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

где $y_1, \ldots, y_s$ – все переменные, использованные в атомах $C_1, \ldots, C_k$, $s \geq 0$, см.[1].

Язык чистого хорновского программирования определяется заданием шестерки: $M, V, \Pi, \Phi, Prog, \Delta$, где M – универсум Эрбрана, построенный из элементов множества Φ, символы же множества Φ, проинтерпретированы описанным выше образом. Множество программ $Prog$ представляет собой непустое подмножество программ, построенных из элементов множеств V, Π, Φ, а отображение Δ, каждой программе $P \in Prog$ сопоставляет непустое подмножество непустых запросов $\Delta(P)$, построенных из элементов множеств V, Π, Φ.

Перед тем как определить правило SLD–резолюции напомним определения подстановки, композиции подстановок, унификатора, наиболее общего унификатора, взятые из [2]. Подстановка есть множество вида $\{t_1/x_1,\ldots,t_n/x_n\}$, где t_i – терм, x_i – переменная, $t_i \neq x_i$, $i \neq j \Rightarrow x_i \neq x_j$, $i=1,\ldots,n$, $n \geq 0$. Если $n=0$ то подстановка называется пустой. Пустую подстановку условимся обозначать ε.

---

[1] *Lloyd J.W.* Foundations of Logic Programming. // Springer-Verlag, 1984, 120p.

[2] *Chang C.L., Lee R.C.T.* Symbolic Logic and Mechanical Theorem Proving. //Academic Press, 1973 (рус. пер. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем. //М.: Наука, 1983, 358 с.).

Выражением назовем атом или терм. Пусть E – выражение и $\sigma=\{t_1/x_1,\ldots,t_n/x_n\}$ – подстановка. Через $E\sigma$ обозначим выражение, полученное из E путем одновременной подстановки термов $t_1,\ldots,t_n$ вместо переменных $x_1,\ldots,x_n$, соответственно. Традиционным образом вводится композиция подстановок, которая является ассоциативной операцией. Пусть $\sigma_1=\{t_1/x_1,\ldots,t_n/x_n\}$ и $\sigma_2=\{\tau_1/y_1,\ldots,\tau_n/y_m\}$, где $n,m\geq 0$, - подстановки. Подстановку $\sigma_3=\sigma_1\sigma_2$ назовем композицией подстановок σ_1 и σ_2, если σ_3 получена из множества $\{t_1\sigma_2/x_1,\ldots,t_n\sigma_2/x_n, \tau_1/y_1,\ldots,\tau_n/y_m\}$, посредством удаления следующих его элементов: если $t_i\sigma_2=x_i$, то удаляется элемент $t_i\sigma_2/x_i$, $i=1,\ldots,n$, если $y_j=\{x_1,\ldots,x_n\}$, то удаляется элемент τ_j/y_j, $j=1,\ldots,$m. Напомним, что для любых подстановок σ, δ, γ и выражения E имеем: $(\sigma\delta)\gamma=\sigma(\delta\gamma)$ и $(E\sigma)\delta=E(\sigma\delta)$.

Будем говорить, что атомы A_1 и A_2 унифицируемы, если существует такая подстановка δ, что $A_1\delta=A_2\delta$. Подстановку δ назовем унификатором A_1 и A_2. Унификатор σ атомов A_1 и A_2, назовем наиболее общим унификатором A_1 и A_2 ($\sigma = mgu(A_1,A_2)$), если для любого их унификатора δ существует подстановка γ такая, что $\sigma\gamma=\delta$.

Определим правило *SLD*–резолюции с функцией выбора первого атома в запросе. Пусть S - предложение $A\text{:-}B_1,\ldots,B_m$, где $m\geq 0$, и Q запрос вида $?\text{-}C_1,\ldots,C_k$, где $k>0$. Правило *SLD*–резолюции применимо к Q и S, если Q и S не имеют общих переменных и C_1 с A унифицируемы, тогда результатом применения правила *SLD*–резолюции к Q и S является запрос $SLD(Q,S)$, имеющий вид:

$$?\text{-}B_1\sigma,\ldots,B_m\sigma,C_2\sigma,\ldots,C_k\sigma, \quad \text{где } \sigma = mgu(C_1,A).$$

Последовательность запросов $Q_1,Q_2,\ldots$ назовем *SLD*–выводом из пары (P,Q), где P программа вида $S_1,\ldots,S_n$, $n>0$, а Q – непустой запрос, если $Q=Q_1$ и $Q_{i+1}=SLD(Q_i,S_j)$, для некоторого $S_j\in\{S_1,\ldots,S_n\}$, $i\geq 1$. *SLD*–вывод может быть как конечным, так и бесконечным. Если $Q_1,\ldots,Q_r$, $r\geq 1$, конечный *SLD*–вывод из пары (P,Q), то будем говорить, что из программы P и запроса Q *SLD*–выводим запрос Q_r, обозначим это $(P,Q)\,|\!-\!Q_r$. Из результатов работы[1] следует теорема 1.2.1.

Теорема 1.2.1. Пусть P – программа и Q – непустой запрос, тогда:

$$P\,|\!=\!Q \;\Leftrightarrow\; (P,Q)\,|\!-\,?\text{-}\,.$$

Раздел 1.3 посвящен ПРОЛОГу. Чистый ПРОЛОГ (далее просто ПРОЛОГ) представляет собой пример языка чистого хорновского программирования, где V – счетное множество предметных переменных, Π содержит счетное число предикатных символов местности n для любого $n\geq 0$, Φ содержит счетное число функциональных символов местности n для любого $n\geq 0$, $Prog$ представляет собой множество всех хорновских программ, построенных из элементов множеств V, Π, Φ, а отображение Δ, каждой программе $P\in Prog$ сопоставляет множество всех непустых хорновских запросов, построенных из элементов множеств V, Π, Φ. Опишем интерпретатор ПРОЛОГа U_0. Пусть P – программа, и Q – непустой запрос.

---

[1] *Lloyd J.W.* Foundations of Logic Programming. // Springer-Verlag, 1984, 120 p.

Шаг 1. Q_1 взять равным Q; $i=1$; опустошить стек; перейти к шагу 2.

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

Шаг 3. Просмотреть предложения программы P согласно их порядку до первого такого предложения $S_к$, что правило *SLD*-резолюции применимо к Q_i и $S_к$. Если такое $S_к$ существует, то $Q_{i+1} = SLD(Q_i,S_к)$; $i=i+1$; k поместить в стек и перейти к шагу 2, иначе перейти к шагу 4.

Шаг 4. Если i равно 1, то остановка с отрицательным ответом, иначе перейти к шагу 5.

Шаг 5. Прочесть верхний символ стека. Пусть он равен d. Просмотреть предложения программы P, начиная с S_{d+1}, согласно их порядку, до первого такого S_j, ($j>d$), что правило *SLD*-резолюции применимо к Q_{i-1} и S_j. Если такое S_j существует, то $Q_i=SLD(Q_{i-1},S_j)$; j поместить в стек и перейти к шагу 2, иначе $i=i-1$ и перейти к шагу 4.

Из результатов работы[1] следует теорема 1.3.1.

Теорема 1.3.1. Интерпретатор ПРОЛОГа U_0 является непротиворечивым и не является логически полным.

В ГЛАВЕ 2 процедурный язык программирования определяется как логический язык программирования, основанный на логике предикатов первого порядка, исследуются процедурная программа и интерпретатор процедурного языка программирования с логической точки зрения. Глава 2 состоит из одного раздела 2.1.

В разделе 2.1 дается несколько упрощённое, не меняющее сути дела, определение процедурного языка прграммирования. Зададимся четыремя непустыми непересекающимися множествами M, V, $B\Pi$, Φ, которые имеют описанный в разделе 1.1 смысл. $V = X \cup Y \cup \{z\}$, где X – множество входных переменных, Y – множество рабочих переменных, z – выходная переменная. Конструкцию вида $u=\tau$, где $u \in Y \cup \{z\}$, τ - терм, имеющий вид $f(t_1, \ldots, t_k)$, v или m, где f – k-местная функция из Φ, $t_i \in M \cup X \cup Y$, $i=1,\ldots,k$, $k>0$, $v \in X \cup Y$, $m \in M$, назовём элементарным оператором, а переменную u его левой частью. Конечное множество элементарных операторов с различными левыми частями назовём оператором. Атом вида $p(t_1,\ldots, t_k)$ или его отрицание $\neg p(t_1,\ldots,t_k)$, где p - k-местный символ из $B\Pi$, $t_i \in M \cup X \cup Y$, $i=1,\ldots,k$, $k>0$, назовём условием.

Программа процедурного языка в графовой форме (г.программа) представляет собой конечный ориентированный граф с одной выделенной вершиной - входом и одной выделенной вершиной - выходом. Из выхода не исходит ни одной дуги, во вход не входит ни одна дуга. Из каждой вершины, отличной от выхода, исходит либо одна дуга, либо – две: если из вершины исходит одна дуга, то она помечена только оператором; если исходит – две, то каждая из них помечена как оператором, так и условием, причём, если одна из них помечена

---

[1] *Lloyd J.W.* Foundations of Logic Programming. // Springer-Verlag, 1984, 120 p.

условием $p(t_1,\ldots, t_k)$, то другая условием $\neg p(t_1,\ldots,t_k)$. Оператор, сопоставляемый дуге, ведущей в выход, имеет вид $z=\tau$, дуге, не ведущей в выход – $\{u_1=\tau_1,\ldots,u_d=\tau_d\}$, $u_i \in Y$, $i=1,\ldots,d$, $d>0$. Пусть $\bar{x}=\langle x_1,\ldots,x_k\rangle$ - вектор входных переменных г.программмы P_g и $\bar{x}_0=\langle x_1^0,\ldots,x_k^0\rangle$, где $x_i^0 \in M$, $i=1,\ldots,k$, $k>0$. Выполнение г.программы P_g на $\bar{x}_0$ начинается со входа и определяется естественным образом. Выполнение P_g на $\bar{x}_0$ может быть как конечным, так и бесконечным. В случае конечного выполнения процесс завершается в выходе и результатом является полученное значение выходной переменной z.

Опишем семерку M, V, Π, $B\Pi$, Φ, $Prog$, Δ, определяющую процедурный язык, как логический язык программирования. Здесь множества M, V, $B\Pi$, Φ являются соответствующими множествами процедурного языка; Π – множество предикатных символов с приписанной каждому символу местностью, причём для любого $k>0$ Π содержит счётное число символов местности k. Множество программ $Prog$ получим из множества г.программ процедурного языка, строя по каждой г.программе P_g логическую программу $P \in Prog$. Опишем алгоритм построения.

Пусть P_g – г.программа, $\bar{x}=\langle x_1,\ldots,x_к\rangle$ - вектор её входных переменных, $\bar{y}=\langle y_1,\ldots,y_h\rangle$ - вектор её рабочих переменных. Пусть P_g имеет $n+2$ $(n\geq 0)$ вершины. Перенумеруем их числами от 0 до $n+1$ следующим образом: входу сопоставим 0, выходу – $n+1$, остальные вершины перенумеруем произвольным образом. Сопоставим каждой дуге (r,s) г.программы P_g формулу $A_{(r,s)}$, построенную из M, V, Π, $B\Pi$, Φ, $0\leq r,s\leq n+1$. Пусть q - $k+1$-местный, а $q_1,\ldots,q_n$ – $k+h$-местные предикатные символы из Π. Введём обозначение: если дуге (r,s), где $0\leq r,s\leq n$, г.программы P_g сопоставлен оператор $\{u_1=\tau_1,\ldots,u_d=\tau_d\}$, $u_i \in Y$, $i=1,\ldots,d$, $d>0$, то атом, полученный в результате подстановки в $q_s(\bar{x},\bar{y})$ вместо каждой переменной u_i терма τ_i $(1\leq i\leq d)$ обозначим $q_s(\bar{x},\bar{\tau}_{(r,s)})$. Определим $A_{(r,s)}$, $0\leq r,s\leq n+1$. $A_{(r,s)}$ будет равна одной из следующих формул:

$q(\bar{x},\tau)$, если $r=0$, $s=n+1$, из входа исходит одна дуга и она помечена оператором $z=\tau$;

$\pi \supset q(\bar{x},\tau)$, если $r=0$, $s=n+1$, из входа исходят две дуги и и дуга $(0, n+1)$ помечена условием π и оператором $z=\tau$;

$q_s(\bar{x},\bar{\tau}_{(r,s)})$, если $r=0$, $s\neq n+1$, из входа исходит одна дуга;

$\pi \supset q_s(\bar{x},\bar{\tau}_{(r,s)})$, если $r=0$, $s\neq n+1$, из входа исходят две дуги и дуга $(0, s)$ помечена условием π;

$q_r(\bar{x},\bar{y}) \supset q(\bar{x},\tau)$, если $r\neq 0$, $s=n+1$, из вершины r исходит одна дуга и она помечена оператором $z=\tau$;

$q_r(\bar{x},\bar{y}) \& \pi \supset q(\bar{x},\tau)$, если $r\neq 0$, $s=n+1$, из вершины r исходят две дуги и дуга $(r, n+1)$ помечена оператором $z=\tau$ и условием π;

$q_r(\bar{x},\bar{y}) \supset q_s(\bar{x},\bar{\tau}_{(r,s)})$, если $r\neq 0$, $s\neq n+1$, из вершины r исходит одна дуга;

$q_r(\bar{x},\bar{y}) \& \pi \supset q_s(\bar{x},\bar{\tau}_{(r,s)})$, если $r\neq 0$, $s\neq n+1$, из вершины r исходят две дуги и дуга (r, s) помечена условием π.

Логическая программа P будет иметь вид:

$$\forall \bar{x}\ \forall \bar{y}\ A(\bar{x},\bar{y}), \qquad (2.1.1)$$

где $A(\bar{x}, \bar{y})$ – конъюнкция формул $A_{(r,s)}$, сопоставленных всем дугам г.программы P_g.

Имея программу $P \in Prog$ можно всегда восстановить г.программу P_g из которой получена программа P, т.к. каждый член $A_{(r,s)}$ конъюнкции $A(\bar{x}, \bar{y})$ есть изображение дуги (r, s) г.программы P_g в виде логической формулы.

Теорема 2.1.1. Всякая программа $P \in Prog$ имеет наименьшую модель.

Пусть $P \in Prog$ и имеет вид (2.1.1). Определим множество запросов $\Delta(P)$, соответствующих программе P.

$$\Delta(P) = \{\exists z\, q(\bar{x}_0, z) \mid \bar{x}_0 = \langle x_1^0, \ldots, x_k^0 \rangle, \text{ где } x_i^0 \in M, i=1,\ldots,k\}$$

Теорема 2.1.2. Пусть $P \in Prog$, $Q \in \Delta(P)$ и имеет вид $\exists z\, q(\bar{x}_0, z)$, тогда:

a) $P \models Q \Leftrightarrow$ существует единственное $z_0 \in M$ такое, что $P \models q(\bar{x}_0, z_0)$;

b) Выполнение P_g на $\bar{x}_0$ конечно и результат равен $z_0 \in M \Leftrightarrow P \models q(\bar{x}_0, z_0)$;

c) $P \models Q \Leftrightarrow$ выполнение P_g на $\bar{x}_0$ конечно.

Определим интерпретатор U. Пусть $P \in Prog$, $Q \in \Delta(P)$ и имеет вид $\exists z\, q(\bar{x}_0, z)$, тогда U на P и Q останавливается с положительным ответом со значением $z_0 \in M$, если выполнение P_g на $\bar{x}_0$ конечно и результат равен z_0; U на P и Q функционирует бесконечно, если выполнение P_g на $\bar{x}_0$ бесконечно.

Из теоремы 2.1.2 и алгоритма интерпретатора U следует теорема 2.1.3.

Теорема 2.1.3. Пусть $P \in Prog$, $Q \in \Delta(P)$ и имеет вид $\exists z\, q(\bar{x}_0, z)$, тогда:

a) Если $U(P,Q) = z_0 \in M$, то $P \models q(\bar{x}_0, z_0)$;

b) Если $P \models Q$, то $U(P,Q) = z_0 \in M$.

Из теоремы 2.1.3 следует теорема 2.1.4.

Теорема 2.1.4. Интерпретатор процедурного языка программирования U является непротиворечивым и логически полным.

В ГЛАВЕ 3 недетерминированные конечные автоматы определяются как язык хорновского программирования. Основным результатом является теорема Теорема 3.1.1 о корректности такого определения. Глава 3 состоит из одного раздела 3.1.

Раздел 3.1. Зафиксируем три непересекающихся множества V, Π и Φ, где V – счетное множество переменных; $\Pi = \Pi_1 \cup \Pi_2 \cup \Pi_3$, где Π_1 – множество 1-местных предикатных символов, Π_2 - множество 2-местных предикатных символов, Π_3 - множество 3-местных предикатных символов:

Π_1={*state, letter, word, initial_state, final_ state, accept*},

Π_2={*nil_transition, accept_from*},

Π_3={*transition*},

$\Phi = States \cup Letters \cup \{nil,\ append\}$ – множество функциональных символов, где *States* и *Letters* – счетные множества *0*-местных функциональных символов, $States \cap Letters = \varnothing$, *nil* - *0*-местный функциональный символ, $nil \notin States \cup Letters$, *append* - 2-местный функциональный символ. Терм *nil* условимся обозначать так же $[\ ]$, а терм $append(t_1,t_2)$ - $[t_1 | t_2]$, где $t_1,\ t_2$ - термы.

Определим множество недетерминированных конечных автоматов *Automata*. Недетерминированный конечный автомат *A* есть пятерка $(\Xi,\ \Sigma,\ \delta,\ q_0,\ F)$, где:

Ξ – конечное множество состояний, $\Xi \subset States$, $\Xi \neq \varnothing$;

Σ - конечный алфавит входных символов, $\Sigma \subset Letters$, $\Sigma \neq \varnothing$;

$\delta : \Xi \times (\Sigma \cup \{nil\}) \rightarrow 2^{\Xi}$ - функция переходов, где 2^{Ξ} - множество всех подмножеств множества Ξ;

q_0 - начальное состояние, $q_0 \in \Xi$;

F – множество заключительных состояний, $F \subset \Xi$.

Определим множество слов $Words_{\Sigma}$:

1. $[\ ] \in Words_{\Sigma}$;

2. $a \in \Sigma,\ \omega \in Words_{\Sigma} \Rightarrow [a | \omega] \in Words_{\Sigma}$.

Пару (q,ω), где $q \in \Xi,\ \omega \in Words_{\Sigma}$ назовем конфигурацией автомата *A*. Будем говорить, что из конфигурации (q,ω) непосредственно выводима конфигурация (q',ω), где $q,\ q' \in \Xi,\ \omega \in Words_{\Sigma}$, и обозначать это $(q,\omega) \rightarrow (q',\omega)$, если $q' \in \delta(q,\ nil)$. Будем говорить, что из конфигурации $(q,\ [a | \omega])$ непосредственно выводима конфигурация (q',ω), где $q,\ q' \in \Xi,\ a \in \Sigma,\ \omega \in Words_{\Sigma}$, и обозначать это $(q,\ [a | \omega]) \rightarrow (q',\omega)$, если $q' \in \delta(q,\ a)$. Будем говорить, что из конфигурации *C* выводима конфигурация *C'* и обозначать это $C \rightarrow\rightarrow C'$, если существует конечная последовательность конфигураций $C_0, C_1, \ldots, C_k$ $(k \geq 0)$ такая, что $C_0 = C$, $C_k = C'$, и $C_i \rightarrow C_{i+1}$, $i = 0,\ldots,k-1$.

Определим язык $L(A)$, $L(A) \subset Words_{\Sigma}$, распознаваемый недетерминированным конечным автоматом *A*:

$$L(A) = \{\omega \mid \omega \in Words_{\Sigma} \text{ и } (q_0,\omega) \rightarrow\rightarrow (q,\ nil), \text{ где } q \in F\}.$$

Опишем хорновскую программу P_A, которую сопоставим недетерминированному конечному автомату *A*. Программа P_A является последовательностью, состоящую из следующих предложений:

если $a \in \Sigma$, то $letter(a) \in P_A$;

если $q \in \Xi$, то $state(q) \in P_A$;

если $q \in F$, то $final\_state(q) \in P_A$;

$initial\_state(q_0) \in P_A$;

если $q' \in \delta(q,a)$, где $q,q' \in \Xi$, $a \in \Sigma$, то $transition(q,a,q') \in P_A$;

если $q' \in \delta(q,nil)$, где $q,q' \in \Xi$, то $nil\_transition(q,q') \in P_A$;

следующие предложения принадлежат P_A для каждого автомата $A \in Automata$:

word([ ])

word([L | W]) :- letter(L), word(W)

accept_from(S,[ ]) :- final_state(S)

accept_from(S,[L|W]) :- state(S), letter(L), word(W), transition(S,L,S1), state(S1),

accept_from(S1,W)

accept_from(S,W) :- state(S), word(W), nil_transition(S,S1), state(S1), accept_from(S1,W)

accept(W) :- word(W), initial_state(S), accept_from(S,W) ,

где $L, S, S1, W \in V$.

Программа P_A называется стандартным представлением автомата A. Очевидно, что два стандартных представления автомата A могут отличаться одно от другого только порядком их предложений.

Логический язык программирования, который соответствует недетерминированным конечным автоматам, представляет собой язык хорновского программирования, определяемый шестеркой $M, V, \Pi, \Phi, Prog, \Delta$, где: M – универсум Эрбрана, построенный из функциональных символов множества Φ; множества V, Π, Φ определены выше; $Prog = \{P_A \mid A \in Automata\}$ есть множество программ, состоящее из всех стандартных представлений всех недетерминированных конечных автоматов множества $Automata$; Δ – отображение, которое по каждой программе $P_A \in Prog$, где $A \in Automata$, определяет множество, соответствующих ей запросов, $\Delta(P_A) = \{?\text{-}accept(\omega) \mid \omega \in Words_\Sigma\}$.

Существенно используя теорему 1.2.1, доказывается теорема 3.1.1.

Теорема 3.1.1. Пусть $A=(\Xi, \Sigma, \delta, q_0, F)$ - недетерминированный конечный автомат, P_A - его некоторое стандартное представление и $\omega \in Words_\Sigma$. Тогда:

$$\omega \in L(A) \Leftrightarrow P_A \models ?\text{-}accept(\omega).$$

В ГЛАВЕ 4 исследуется интерпретатор системы ПРОЛОГ U_0, как интерпретатор хорновского языка программирования, соответствующего недетерминированным конечным автоматам. Глава 4 состоит из двух разделов 4.1-4.2.

Разделе 4.1 посвящен недетерминированным конечным автоматам и интерпретатору системы ПРОЛОГ.

Теорема 4.1.1. a) Существует такой недетерминированный конечный автомат A и такое $\omega \in L(A)$, что для любого стандартного представления P_A будем иметь: $U_0(P_A, ?\text{-}accept(\omega))$ не определено.

b) Существует такой недетерминированный конечный автомат A и такое $\omega \notin L(A)$, что для любого стандартного представления P_A будем иметь: $U_0(P_A, ?\text{-}accept(\omega))$ не определено.

Стандартное представление P_{Aopt} недетерминированного конечного автомата $A=(\Xi, \Sigma, \delta, q_0, F)$ назовем оптимальным стандартным представлением для U_0, если для любого стандартного представления P_A автомата A и любого $\omega \in Words_\Sigma$ имеем: $U_0(P_A, ?\text{-}accept(\omega))$ определено $\Rightarrow U_0(P_{Aopt}, ?\text{-}accept(\omega))$ определено.

Теорема 4.1.2. Существует недетерминированный конечный автомат A, не имеющий оптимального стандартного представления для интерпретатора U_0.

Раздел 4.2 посвящен детерминированным конечным автоматам и интерпретатору системы ПРОЛОГ. Недетерминированный конечный автомат $A=(\Xi, \Sigma, \delta, q_0, F)$ называется детерминированным конечным автоматом, если:

1. для любых $q\in \Xi$ и $a\in\Sigma\cup\{nil\}$ имеем: мощность множества $\delta(q, a) \leq 1$;
2. для любого $q\in \Xi$ имеем: $\delta(q, nil)\neq\varnothing \Rightarrow$ для любого $a\in\Sigma$, $\delta(q, a)=\varnothing$.

Теорема 4.2.1. a) Существует такой детерминированный конечный автомат A, такое $\omega\in L(A)$, и такое его стандартное представление P_A, что $U_0(P_A, ?\text{-}accept(\omega))$ не определено.

b) Существует такой детерминированный конечный автомат A и такое $\omega\notin L(A)$, что для любого стандартного представления P_A будем иметь: $U_0(P_A, ?\text{-}accept(\omega))$ не определено.

Теорема 4.2.2. Всякий детерминированный конечный автомат A имеет оптимальное стандартное представление P_{Aopt} для U_0 и, если $\omega\in L(A)$, то $U_0(P_{Aopt}, ?\text{-}accept(\omega))$ определено.

Недетерминированный конечный автомат $A=(\Xi, \Sigma, \delta, q_0, F)$ называется конечным автоматом, если:

1. для любых $q\in \Xi$ и $a\in\Sigma$ имеем: мощность множества $\delta(q, a)$ равна 1;
2. для любого $q\in \Xi$ имеем: $\delta(q, nil)=\varnothing$.

Теорема 4.2.3. Пусть $A=(\Xi, \Sigma, \delta, q_0, F)$ - конечный автомат. Тогда для любого его стандартного представления P_A и любого $\omega\in Words_\Sigma$, имеем: $U_0(P_A, ?\text{-}accept(\omega))$ определено.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ

1. Дается формальное определение логического языка программирования, основанного на логике предикатов первого порядка. Как пример логических языков программирования, основанных на логике предикатов первого порядка, рассматриваются языки хорновского программирования. Обсуждается язык хорновского программирования ПРОЛОГ и его интерпретатор U_0.
2. Процедурный язык программирования определяется как логический язык программирования, основанный на логике предикатов первого порядка. Доказывается коррекность такого определения. Доказывается также, что всякая процедурная программа имеет наименьшую модель и что интерпретатор процедурного языка программирования является логически полным и непротиворечивым.

3. Недетерминированные конечные автоматы определяются как язык хорновского программирования. Доказывается корректность такого определения.
4. Доказывается неполнота интерпретатора U_0 в случае языка хорновского программирования, соответствующего недетерминированным конечным автоматам. Доказывается также, что: существует недетерминированный конечный автомат, не имеющий оптимального для U_0 стандартного представления; каждый детерминированный конечный автомат имеет оптимальное для U_0 стандартное представление; всякое стандартное представление конечного автомата является оптимальным для U_0.

ПУБЛИКАЦИИ ПО ТЕМЕ ДИССЕРТАЦИИ

1. *Нигиян С.А., Хачоян Л.О., Нигиян А.В.* К логической трактовке процедурного программирования. // Доклады НАН Армении, том 107, №1, 2007, с. 20-25.

2. *Nigiyan A.V*. Nondeterministic Finite Automata from the Logical Standpoint. //Proceedings of the Conference on Computer Science and Information Technologies (CSIT-2007), Yerevan, 2007, p. 58-59.

3. *Нигиян С.А., Нигиян А.В.* К логической трактовке недетерминированных конечных автоматов. // Доклады НАН Армении, том 108, №2, 2008, с. 124 - 132.

ԱՄՓՈՓԱԳԻՐ

Արամ Վիգենի Նիգիյան

«Ծրագրավորման պրոցեդուրային լեզուների և ոչ դետերմինացված վերջավոր ավտոմատների տրամաբանական մոդելավորում»

Ատենախոսության հիմնական արդյունքներն են.

1. Տրվում է առաջին կարգի պրեդիկատների տրամաբանության վրա հիմնված տրամաբանական ծրագրավորման լեզվի ֆորմալ սահմանումը: Որպես առաջին կարգի պրեդիկատների տրամաբանության վրա հիմնված տրամաբանական ծրագրավորման լեզվի օրինակ դիտարկվում են Հորնի ծրագրավորման լեզուները: Քննարկվում է Հորնի ծրագրավորման ՊՐՈԼՈԳ լեզուն և նրա U_0 ինտերպրետատորը:
2. Պրոցեդուրային ծրագրավորման լեզուն սահմանվում է որպես առաջին կարգի պրեդիկատների տրամաբանության վրա հիմնված տրամաբանական ծրագրավորման լեզու: Ապացուցվում է այդպիսի սահմանման կոռեկտությունը: Ապացուցվում է նաև, որ ցանկացած պրոցեդուրային ծրագիր ունի փոքրագույն

մոդել և որ պրոցեդուրային ծրագրավորման լեզվի ինտերպրետատորը լրիվ է ու անհակասելի:

3. Ոչ դետերմինացված վերջավոր ավտոմատները սահմանվում են որպես Հորնի ծրագրավորման լեզու: Ապացուցվում է այդպիսի սահմանման կոռեկտությունը:
4. Ապացուցվում է U_0 ինտերպրետատորի ոչ լրիվությունը ոչ դետերմինացված վերջավոր ավտոմատներին համապատասխանող Հորնի ծրագրավորման լեզվի դեպքում: Ապացուցվում է նաև, որ գոյություն ունի ոչ դետերմինացված վերջավոր ավտոմատ, որը չունի օպտիմալ ստանդարտ ներկայացում U_0–ի համար, ցանկացած դետերմինացված վերջավոր ավտոմատ ունի U_0–ի համար օպտիմալ ստանդարտ ներկայացում և վերջավոր ավտոմատի ամեն մի ստանդարտ ներկայացում օպտիմալ է U_0–ի համար:

Aram V. Nigiyan

"Logical Modelling of Procedural Programming Languages and Nondeterministic Finite Automata"