ЛОГИЧЕСКИЕ МЕТОДЫ В ПРОГРАММИРОВАНИИ
(Вычислительные системы)
1987 год Выпуск 120

УДК 519.68

СЕМАНТИКА ПАРАМЕТРИЧЕСКИХ КОНСТРУКЦИЙ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

В.Б. Борщёв

§1. Предварительные сведения и необходимые понятия

В логическом программировании (языки типа ПРОЛОГ) уже давно обсуждается идея использования конструкций с параметрами, аналогичными используемым в традиционных языках параметрам типа процедур, либо параметрическим конструкциям в абстрактных типах данных (см., например, [1-4]). Параметры чаще всего вводятся при модульном построении программ. Использование параметров позволяет повышать универсальность модулей (их reusability), хотя механизмы, обеспечивающие сборку программы из модулей, и механизмы параметризации достаточно независимы (тем не менее они должны естественным образом сочетаться при модульном построении программ).

В настоящей работе обсуждается только параметризация и в первую очередь теоретико-модельная семантика параметрических конструкций. Основной мотив логического стиля программирования заключается в том, что программист, составляя программу, прежде всего видит ее теоретико-модельную семантику. Между тем семантика параметрических конструкций, как правило, четко не описывалась.

Рассмотрим примеры. Программами

```
Listnat(nil) ←
Listnat(x.y) ← Nat(x), Listnat(y)
```

и

```
Listeven(nil) ←
Listeven(x.y) ← Even(x), Listeven(y)
```

задаются соответственно списки натуральных и четных чисел (в предположении, что символами Nat и Even задаются натуральные и четные числа - соответствующие правила приведены в §2). Программы эти очень похожи, и,естественно, возникает желание написать еще одну параметрическую программу, "частными случаями" которой были бы как эти программы, так и, например, программа, задающая списки списков натуральных чисел и т.д.

Обычно предлагают программы типа[x)]

```
List(nil,p) ←
List(x.y,p) ← p(x), List(y,p).                    (I)
```

Однако теоретико-модельная и процедурная семантики таких программ не описываются, хотя обычно указывается, что они лежат за пределами языка первого порядка. По-видимому, имеется в виду, что p - "предикатная" переменная, значением которой может быть произвольное унарное отношение (например, отношения Nat, Even и т.д.), а List - бинарное отношение "второго порядка", связывающее "обычные" объекты (списки) с отношениями. Поэтому в запросах и других правилах можно употреблять атомы List(y,Nat), List(Suc(Suc(0)), Even) и т.д.

В работе [2] обсуждается другой подход к составлению таких программ: там параметрические конструкции предлагается рассматривать как "синтаксический сахар", скрывающий обычные логические программы, в частности для программы (I) - программу

```
List(nil,p) ←
List(x.y,p) ← apply(p,x), List(y,p),              (2)
```

в которой p - уже обычная переменная. Недостаток этого решения заключается в том, что оно представляет некоторую двусмысленность (double-think): с одной стороны, о программах можно думать как о параметрических, с другой - выполнять их можно как обычные программы. Кроме того, возникают и некоторые технические трудности. Как, скажем, с помощью такой программы определить списки списков натуральных чисел? Предлагаемым для этой цели атомам типа List(y, List(Nat)) уже трудно сопоставить ясную семантику (в рамках "обычного" логического программирования). Эти трудности, правда, можно легко избежать при более последовательном применении символа apply (см. §4).

Ниже предлагаются конструкции, имеющие простую теоретико-модельную семантику, которая естественным образом связана с процедурной. Для данного примера наша программа будет иметь вид:

---

x) Такая программа приводится в [2] (в ней вместо символа List используется символ have-property). Аналогичные программы рассматриваются в [I] и [4].

$$\text{List}[p](\text{nil}) \leftarrow \qquad (3)$$

$$\text{List}[p](x.y) \leftarrow p(x),\ \text{List}[p](y)\ .$$

Здесь p - параметр, значением которого могут быть любые унарные символы, а List[p] - сложный символ. Делая подстановки (заменяя p), можно получить из него символы List[Nat] , List[Even], List[List[p]] и т.д. и использовать такого рода символы в других правилах и запросах. Так, правило

$$\text{MList}(x.y) \leftarrow \text{List}[\text{Nat}](y),\ \text{Length}(y,x)$$

определяет списки натуральных чисел y, первый элемент которых x является длиной "хвоста" (в этом правиле Length - отношение между списком y и числом его элементов x). Наши правила интерпретируются стандартным образом на моделях в бесконечной сигнатуре, состоящей из символов типа Nat , Even , List[Nat] , List [List [Even]] и т.д.

Предполагается, что читатель знаком с основными понятиями логического программирования [5-8], однако для полноты изложения в §2 кратко описаны синтаксис и семантика "обычных" логических программ; §3 посвящен собственно параметрическим конструкциям. Поскольку существует много версий логического программирования и параметрические конструкции можно ввести для любой из них,то в данной работе мы их вводим для классической версии (в [8] они рассматриваются для так называемой однородной версии). В §4 обсуждается представление параметрических программ в виде обычных логических программ.

В статье используются следующие сокращения: ≑ - равенство по определению и ⇒ - "влечет".

§2. Обычные логические программы

Логическая программа описывает "мир задачи" в виде множества объектов, на котором определяются некоторые отношения. Для представления объектов служат термы. Программа состоит из набора утверждений, истинных в этом мире. Утверждения называются правилами. Правила - это специального вида формулы языка первого порядка.

Синтаксис. Логическая программа - это множество правил. Правило имеет вид $A_0 \leftarrow A_1,\dots,A_m$, $m \geq 0$, где $A_0,A_1,\dots,A_m$ - атомы. Левая часть правила (A_0) называется заголовком, а правая $(A_1,\dots$

$\ldots, A_m$) – телом. Если тело пусто ($m = 0$), то правило называется фактом. Атом (атомарная формула) имеет вид $\omega(t_1, \ldots, t_n)$, где ω – n-арный предикатный символ, а $t_1, \ldots, t_n$ – термы. Терм – это либо переменная, либо константа, либо составной терм вида $\sigma(t_1, \ldots, t_l)$, где σ – l-арный функциональный символ, а $t_1, \ldots, t_l$ – термы (константы понимаются как 0-арные функциональные символы). Запрос имеет вид $\leftarrow C_1, \ldots, C_r$, $r \geq 0$, где C_i – атом.

ПРИМЕР. Натуральные числа можно представить в виде термов, построенных из константы 0 и унарного функционального символа Suc: ноль – 0, I – Suc(0), 2 – Suc(Suc(0)) и т.д. С помощью константы nil и бинарного функционального символа "." (точка) строятся списки; nil представляет пустой список, а точка служит для образования нового списка x.y из произвольного объекта x и списка y (точка по традиции употребляется инфиксно вместо стандартной префиксной записи .(x,y)). Так, терм 0.Suc(0).0.nil представляет список [0,I,0].

Правилами

Nat(0) ←

Nat(Suc(x)) ← Nat(x)

Even(0) ←

Even(Suc(Suc(x)) ← Even(x),

образованными с помощью унарных предикатных символов Nat и Even, выделяются из множества всех объектов множества натуральных и четных чисел соответственно, а приведенными во введении правилами для символов Listnat и Listeven определяются множества списков натуральных и четных чисел.

Правилами

Plus(0,x,x) ← Nat(x)

Plus(Suc(x),y,Suc(z)) ← Plus(x,y,z)

определяется сложение на множестве натуральных чисел (с помощью тернарного предикатного символа Plus).

Заметим, что в логическом программировании (по крайней мере, в его "классической" версии) функциональные символы используются только для конструирования объектов (причем разные термы представляют разные объекты), а "настоящие" функции (типа сложения в нашем примере) представляются в виде отношений.

Примеры запросов: $\leftarrow Plus(0, Suc(0), x)$, $\leftarrow Plus(x, y, Suc(Suc(0))$ ("чему равна сумма 0 и 1?", "сумма каких чисел x и y равна 2?").

Подстановки. Теоретико-модельная семантика. В логической программе S и множестве запросов к ней фиксируются использованные в них множество Ω предикатных символов и множество Σ функциональных символов (предикатная и функциональная сигнатуры). Фиксируем для удобства также множество переменных X. Тогда определены множества всех термов $F_\Sigma(X)$ (свободная алгебра в сигнатуре Σ со множеством свободных образующих X) и $I_\Sigma \subset F_\Sigma(X)$ (инициальная алгебра без переменных).

Подстановкой называется отображение $\theta: X \to F_\Sigma(X)$. Оно естественным образом продолжается на термы ($\theta a \leftrightharpoons a$ для каждой константы $a \in \Sigma$ и $\theta t \leftrightharpoons \sigma(\theta t_1, \ldots, \theta t_l)$, для составного терма $t = \sigma(t_1, \ldots, t_l)$) и на атомы ($\theta A \leftrightharpoons \omega(\theta t_1, \ldots, \theta t_n)$ для $A = \omega(t_1, \ldots, t_n)$), а тем самым – на правила и запросы (правилу $A_0 \leftarrow A_1, \ldots, A_m$ ставится в соответствие правило $\theta A_0 \leftarrow \theta A_1, \ldots, \theta A_m$, а запросу $\leftarrow C_1, \ldots, C_r$ – запрос $\leftarrow \theta C_1, \ldots, \theta C_m$). Содержательно подстановкой θ заменяется в каждом выражении e каждое вхождение переменной x на терм θx. Для подстановок естественным образом определяются их композиции: $(\theta_1\theta_2)e \leftrightharpoons \theta_1(\theta_2 e)$.

Образы θr и θQ, сопоставляемые подстановкой θ правилу r и запросу Q, называются их частными случаями. Если же при этом подстановка θ является перестановкой $X \to X$, то частные случаи соответствующих конструкций называются их вариантами.

Программа задает отношения на множестве I_Σ всех объектов (так называемом эрбрановом универсуме). Отношение, сопоставляемое символу $\omega \in \Omega$, отождествляется с некоторым множеством атомов вида $\omega(t_1, \ldots, t_n)$, $t_i \in I_\Sigma$.

Произвольное множество атомов без переменных называется интерпретацией. Интерпретация Int называется моделью программы S, если для каждого правила $A_0 \leftarrow A_1, \ldots, A_m$ из S и каждой подстановки $\theta: X \to I_\Sigma$ имеет место

$$\{\theta A_1, \ldots, \theta A_m\} \subseteq Int \to \theta A_0 \in Int. \qquad (4)$$

Для фактов условие (4) вырождается в требование $\theta A_0 \in Int$.

Обозначим через $Mod(S)$ класс всех моделей программы S. Нетрудно видеть, что интерпретация $Int_S \leftrightharpoons \cap Mod(S)$ – пересечение всех моделей программы S – также является моделью S (если условие (4) выполняется в каждой модели S, то оно выполняется в Int_S). Назовем Int_S главной моделью программы S.

Семантика запросов. Пусть $x_1,\ldots,x_k$ - все переменные запроса $\leftarrow C_1,\ldots,C_r$ к программе S и для некоторой подстановки θ имеет место $\{\theta C_1,\ldots,\theta C_r\}\subseteq \mathrm{Int}_S$. Тогда кортеж термов $\langle t_1,\ldots, t_k\rangle$, где $t_i = \theta x_i$, называется ответом на запрос $\leftarrow C_1,\ldots, C_r$. Множество всех ответов на запрос - это k-арное отношение на множестве I_Σ. Если $k = 0$, т.е. в запросе нет переменных, то ответом будет "да", когда $\{C_1,\ldots,C_r\} \subseteq \mathrm{Int}_S$, и "нет" - в противном случае.

Процедурная семантика. Унификация. Подстановка θ называется унификатором атомов A и A', если $\theta A = \theta A'$. Унификатор θ называется самым общим для A и A', если для любого другого унификатора θ' этих атомов найдется подстановка θ'', такая, что $\theta' = \theta''\theta$.

Шаг вычислений. Пусть Q - запрос вида $\leftarrow C_1,\ldots, C_{i-1},C_i, C_{i+1},\ldots, C_r$ и $A_0 \leftarrow A_1,\ldots, A_m$ - вариант некоторого правила из S, такой,что все его переменные отличны от переменных запроса.Если θ - самый общий унификатор для C_i и A_0, то говорят,что запрос Q' вида $\leftarrow \theta C_1,\ldots, \theta C_{i-1}, \theta A_1,\ldots,\theta A_m, \theta C_{i+1},\ldots, \theta C_r$ выводим из запроса Q с помощью правила $A_0\leftarrow A_1,\ldots,A_m$ и подстановки θ.

Вычисление. Вычислением запроса Q называется последовательность (может быть, бесконечная) запросов $\alpha = Q_1,\ldots,Q_n,\ldots$, такая, что $Q_1 = Q$ и Q_{i+1} выводим из Q_i. Если Q_n - пустой запрос, то вычисление называется успешным. С ним связывается подстановка $\theta_\alpha = \theta_{n-1}\theta_{n-2}\ldots \theta_1$, где θ_i - подстановка,с помощью которой запрос Q_{i+1} выводим из Q_i. Если $x_1,\ldots,x_k$ - все переменные запроса Q ,то кортеж термов $\langle t_1,\ldots,t_k\rangle$, $t_i = \theta_\alpha x_i$, называется ответом, выдаваемым этим вычислением (если запрос не содержит переменных, то успешное вычисление выдает ответ "да").

ПРЕДЛОЖЕНИЕ I. Каждый не содержащий переменных частный случай ответа, выдаваемого успешным вычислением на запрос Q, является ответом на этот запрос, и обратно, каждый ответ на запрос является частным случаем ответа на этот запрос, выдаваемого успешным вычислением.

Это предложение (доказательство которого можно найти в [5-8]) говорит о том, что процедурная семантика корректна и полна с точки зрения теоретико-модельной семантики. В результате успешного вычисления мы получим либо ответ, правильный с точки зрения теоретико-модельной семантики, либо ответ с переменными - в этом случае любая подстановка, "устраняющая" переменные, делает этот ответ правильным. С другой стороны, каждый правильный ответ может быть получен в результате успешного вычисления либо непосредственно, либо после подстановки.

§3. Параметрические программы

Пояснения. Наши параметрические программы отличаются от обычных устройством сигнатуры, состоящей из сложных символов. Так, символы List[Nat] и List[List[Nat]] обозначают отношения, являющиеся применением функции List к отношениям Nat и List[Nat] соответственно. "Обычные" символы Nat, Even и т.п. можно понимать, как имена нульарных функций (значениями которых являются соответствующие отношения).

Сигнатурная алгебра. Пусть задана исходная сигнатура Ω, которую теперь (в отличие от рассматриваемой в §2) будем понимать как множество имен функций, причем каждому символу $\omega \in \Omega$ поставлена в соответствие его характеристика (тип) $ch(\omega)$. Характеристика состоит из размерности - кортежа натуральных чисел $\langle i_1,\ldots$ $\ldots, i_k\rangle$, $k \geq 0$, и арности - натурального числа j, в записи - $ch(\omega) = [i_1,\ldots,i_k]\,(j)$. Фиксируем множество P параметров, причем каждому параметру также поставим в соответствие арность j.

Определим по индукции множество $F_\Omega(P)$ сигнатурных термов, каждому из которых также поставим в соответствие арность:

1) параметр $p \in P$ арности j есть сигнатурный терм арности j;

2) если $\omega \in \Omega$ имеет характеристику $ch(\omega) = [i_1,\ldots,i_k]\,(j)$, $k \geq 0$, и $\tau_1,\ldots,\tau_k$ - сигнатурные термы арности $i_1,\ldots,i_k$ соответственно то $\omega[\tau_1,\ldots,\tau_k]$ - сигнатурный терм арности j.

Таким образом, множество $F_\Omega(P)$ сигнатурных термов является свободной многоосновной алгеброй с типизированными посредством характеристик операциями из Ω и свободными образующими из P (каждое основание алгебры $F_\Omega(P)$ состоит из сигнатурных термов одинаковой арности). Назовем $F_\Omega(P)$ сигнатурной алгеброй. Подалгебра $I_\Omega \subset F_\Omega(P)$ сигнатурной алгебры, состоящая из всех сигнатурных термов без параметров, является инициальной алгеброй.

Отображение $\rho: P \to F_\Omega(P)$, такое, что арности термов p и ρp совпадают, назовем параметрической подстановкой. Параметрическая подстановка естественным образом продолжается до гомоморфизма $\rho: F_\Omega(P) \to F_\Omega(P)$ - для каждого терма $\tau = \omega[\tau_1,\ldots,\tau_k]$ положим $\rho\tau \leftrightharpoons \omega[\rho\tau_1,\ldots,\rho\tau_k]$.

Алгебра $F_\Omega(P)$ и будет сигнатурой, в которой строятся правила параметрической программы. Во всем остальном параметрические конструкции не отличаются от конструкций "обычных" логических программ.

ПРИМЕР. Для нашего примера исходная сигнатура содержит рассматривавшиеся ранее символы Nat и Even (с характеристикой [ ](0)), а также символ List с характеристикой [1](1). Пусть p – унарный параметр, тогда p, Nat, Even, List[p], List[Nat], List[List[p]] – примеры сигнатурных термов.

Параметрические программы. Как и в §2, фиксируем множество переменных X, функциональную сигнатуру Σ и тем самым множества "обычных" термов $F_\Sigma(X)$ и I_Σ. Параметрические программы отличаются от обычных только тем, что в качестве предикатных символов используются термы из $F_\Omega(P)$: параметрическое правило имеет вид: $A_0 \leftarrow A_1,\ldots,A_m$, $m \geq 0$, где A_i – атомарная формула (атом) вида $\tau(t_1,\ldots,t_n)$, $t_i \in F_\Sigma(X)$, и τ – n-арный терм из $F_\Omega(P)$. Параметрическая программа – это множество параметрических правил. Запрос к параметрической программе имеет вид $\leftarrow C_1,\ldots,C_r$, $r \geq 0$, где C_i – атом вида $\tau(t_1,\ldots,t_n)$, причем τ – n-арный терм из I_Ω (т.е. запросы не содержат параметров).

Теоретико-модельная семантика. Параметрическая программа так же, как и обычная, задает отношения на эрбрановом универсуме I_Σ, только теперь это, вообще говоря, бесконечный набор отношений, сопоставляемых сигнатурным термам из I_Ω.

Произвольное множество атомов без переменных и без параметров, т.е. атомов вида $\tau(t_1,\ldots,t_n)$, где τ – n-арный терм из I_Ω, а $t_i \in I_\Sigma$, называется интерпретацией параметрической программы. Интерпретация Int называется моделью параметрической программы S, если для каждого правила $A_0 \leftarrow A_1,\ldots,A_m$ из S и каждых подстановок $\theta: X \to I_\Sigma$ и $\rho: P \to I_\Omega$ имеет место

$$\{\rho\theta A_1,\ldots,\rho\theta A_m\} \subseteq \mathrm{Int} \Rightarrow \rho\theta A_0 \in \mathrm{Int}.$$

Определение модели программы S можно было бы дать в "двухступенчатой" форме. Пусть $A_0 \leftarrow A_1,\ldots,A_m$ – правило из S и $\rho: P \to I_\Omega$ – параметрическая подстановка, "устраняющая" параметры. Тогда правило $\rho A_0 \leftarrow \rho A_1,\ldots,\rho A_m$ назовем простым случаем правила $A_0 \leftarrow A_1,\ldots,A_m$. Поставим в соответствие параметрической программе S программу

$$S(I_\Omega) \leftrightharpoons \{ r' \mid r' \text{ – простой случай правила } r \in S\}.$$

Теперь, отвлекаясь от алгебраической структуры I_{Ω}, можно рассматривать ее элементы как символы обычной сигнатуры. Тогда теоретико-модельная семантика $S(I_{\Omega})$ – в общем случае бесконечной программы в бесконечной сигнатуре – задается соответствующим определением для обычной программы (из §2). Моделями программы S по определению считаются модели программы $S(I_{\Omega})$. Нетрудно видеть, что эти два определения эквивалентны.

Как и для обычных программ, в классе $Mod(S)$ всех моделей параметрической программы S существует наименьшая (по отношению $\subseteq$) модель $Int_S \leftrightharpoons \cap Mod(S)$, которую мы также будем называть главной моделью S.

Так как запросы к параметрической программе не содержат параметров, то их теоретико-модельная и процедурная семантики описываются определениями из §2, а предложение I связывает теоретико-модельную семантику с процедурной.

§4. Представление параметрических программ с помощью обычных

Будем говорить, что параметрические программы S и S' эквивалентны, если $Int_S = Int_{S'}$. Будем говорить также, что S и S' эквивалентны относительно запроса Q, если каждый ответ на Q в S является ответом на Q в S', и наоборот.

Каждая параметрическая программа S эквивалентна по определению "обычной" программе $S(I_{\Omega})$, вообще говоря, бесконечной. Возникает вопрос: не существует ли для каждой параметрической программы S эквивалентная ей конечная программа $S' \subset S(I_{\Omega})$, т.е. не являются ли параметрические программы "сокращениями" для обычных программ?

Назовем программу S неограниченной, если не существует эквивалентной ей конечной программы.

ПРЕДЛОЖЕНИЕ 2. С у щ е с т в у ю т н е о г р а н и ч е н н ы е п а р а м е т р и ч е с к и е п р о г р а м м ы.

Примером такой программы является программа (3) для символа List[p] вместе с программами для символов Nat и Even из §2. Однако для каждого запроса Q к этой программе существует конечная программа S' в сигнатуре I_{Ω}, эквивалентная ей относительно Q.

Назовем программу S локально неограниченной, если для некоторого запроса Q к S не существует конечной программы $S' \subset S(I_{\Omega})$, эквивалентной S относительно Q.

ПРЕДЛОЖЕНИЕ 3. Существуют локально неограниченные параметрические программы.

Приведем пример такой программы.

$Zero(0) \leftarrow$

$\omega[p]\ (\langle x,x\rangle) \leftarrow p(x)$,

$\omega[p]\ (\langle\langle x,x,x\rangle\rangle) \leftarrow \omega[\omega[p]](x)$.

В этой программе $\langle\ \rangle$ и $\langle\langle\ \rangle\rangle$ - функциональные символы для пары и тройки соответственно, а ω - функция с характеристикой [1] (I). Нетрудно видеть, что не существует конечной программы без параметров в сигнатуре I_Ω (где $\Omega = \{Zero, \omega\}$), эквивалентной данной программе относительно запроса $\leftarrow \omega[Zero](x)$.

Предложения 2 и 3 не противоречат, конечно, тезису Черча о том, что всегда с помощью подходящей кодировки можно представить в бесконечной сигнатуре модель, задаваемую параметрической программой, в виде обычной модели и построить обычную программу, главной моделью которой будет эта кодирующая модель. Смысл предложений 2 и 3 в том, что это сделать невозможно без кодировки.

Одна из таких кодировок хорошо известна - нужно все сигнатурные термы считать обычными (а параметры - обычными переменными) и вместо каждого n-арного предикатного символа использовать специальный n+1-арный предикатный символ, например квадратные скобки (на каждую арность - свои скобки). Тогда каждый атом $\tau(t_1,\ldots,t_n)$ будет иметь вид $[\tau, t_1,\ldots,t_n]$, а программа (3) будет выглядеть так:

$[List(p), nil] \leftarrow$

$[List(p), x.y] \leftarrow [p,x], [List(p), y]$.

Этот прием используется в работе [2] (где роль скобок играет символ apply), однако там это делается только для некоторых атомов, что и приводит к указанным трудностям. В работе [9] используются только такие предикатные символы (там они изображаются круглыми скобками).

Автор признателен М.Хомякову за многочисленные и полезные беседы, а также А.Ломпу за плодотворное обсуждение параметрических конструкций в стандартном ПРОЛОГе.

Литература

I. KOWALSKI R. The use of metalanguage to assemble object level programs and abstract programs// Proc. of the first intern.logic programming conf. Marseille,1982.

2. WARREN D.H. Higher-order extentions to PROLOG - Are they needed?//Machine Intelligence 10, Ellis and Horwood,1981.

3. GALLAIR H. A stady of PROLOG//Computer program synthesis methodologies: Proc.of the NATO advanced study institute, Reidel, 1983.

4. GOGUEN J.A.,MESEGUER J. Equality, types modules and (why not?) generice for logic programming// J.Logic Programming.-1984. - V.1,N 2.- P.179-210.

5. Van EMDEN M.H., KOWALSKI R. The semantics of predicate logic as a programming language// JACM.- 1976.-V.23,N 4.-P.733-743.

6. LLOYD J.W. Foundation of logic programming.- Springer-Verlag,1984.

7. БОРЩЕВ В.Б. ПРОЛОГ - основные идеи и конструкции //Прикладная информатика. - 1986. - №2. - С. 49-76.

8. БОРЩЕВ В.Б. Логическое программирование //Техническая кибернетика. - 1986. - №2. - С. 89-109.

9. COLMERAUER A., KANOUI H., van CANEGHEM. Prolog, bases theoretiques et development actuels// Tecniques et Science Informatique.-1983.-N 4.- P.277-311.

Поступила в ред.-изд.отд.
I апреля 1987 года