

# Математическая логика и логическое программирование

mk.cs.msu.ru → Лекционные курсы  
→ Математическая логика и логическое программирование (3-й поток)

## Блок 45

Логические программы:  
управление вычислениями,  
оператор отсечения

Лектор:  
**Подымов Владислав Васильевич**  
E-mail:  
**valdus@yandex.ru**

ВМК МГУ, 2023/2024, осенний семестр

# Управление вычислениями

Два основных способа управления вычислениями логических программ, доступные программисту *согласно тому, что уже рассказано про логические программы*:

## 1. Выбор порядка программных утверждений

- ▶ Сначала записывать базу индукции (рекурсии), а затем выполнять индуктивный переход (рекурсивный вызов)
- ▶ Сначала решать задачи простыми способами, а при неуспехе переходить к трудным

## 2. Выбор порядка атомов в телах правил

- ▶ Сначала решать простые подзадачи, а после них сложные
- ▶ Сначала вычислять, и после этого использовать вычисленное

Но этого бывает недостаточно

# Управление вычислениями

Например, вычисление программы

1 : elem(X, X.L);      2 : elem(X, Y.L)  $\leftarrow$  elem(X, L);

согласно стандартной стратегии на запросе ?elem(0, 0.1.0.0.nil)  
(«правда ли, что 0 содержится в последовательности [0, 1, 0, 0]»)  
устроено так (по строкам сверху вниз, в строке слева направо):

?elem(0, 0.1.0.0.nil)  $\xrightarrow{1, \{X'/0, L'/1.0.0.nil\}}$   $\square$  Ответ: да ( $\epsilon$ )

2, {X'/0, Y'/0, L'/1.0.0.nil}  $\downarrow$

?elem(0, 1.0.0.nil)

2, {X'/0, Y'/1, L'/0.0.nil}  $\downarrow$

?elem(0, 0.0.nil)  $\xrightarrow{1, \{X'/0, L'/0.nil\}}$   $\square$  Ответ: да ( $\epsilon$ )

2, {X'/0, Y'/0, L'/0.nil}  $\downarrow$

?elem(0, 0.nil)  $\xrightarrow{1, \{X'/0, L'/0.nil\}}$   $\square$  Ответ: да ( $\epsilon$ )

2, {X'/0, Y'/0, L'/nil}  $\downarrow$

?elem(0, nil)

тупик

А нельзя ли сделать так, чтобы обход завершился после первого «да»?

# Оператор отсечения (!)

Оператор отсечения (!) — это встроенный 0-местный предикат (записывающийся без аргументов и без скобок), предназначенный для особого «отсекания» ветвей дерева вычислений логической программы при использовании стандартной стратегии вычислений

В декларативной семантике ! — это тавтология, формула  $\top$

Как встроенный предикат ! всегда выполняется, и всегда с унификатором  $\epsilon$

В операционную семантику оператором ! вносятся и другие изменения

# Оператор отсечения (!) и стековые вычисления

В терминах **стековых вычислений** оператор **!** выполняется так:

- ▶ Элемент стека может быть помечен произвольным числом **меток отсечения** вида  $!_i$ , где  $i \in \mathbb{N}$
- ▶ При добавлении элемента  $v'$  в стек с головой  $v$  согласно шагу вычисления  $Q \rightarrow Q'$ :
  - ▶ По умолчанию при добавлении элемента в стек все метки  $!_i$  копируются из текущей головы стека
  - ▶ Если  $Q = ?!, B_1, \dots, B_k$ , то из  $v'$  удаляется метка  $!_i$  с наибольшим индексом  $i$
  - ▶ Если  $Q \xrightarrow{\mathcal{R}} Q'$  для правила  $\mathcal{R}$ , в теле которого содержится  $!$ , то в вершины  $v$  и  $v'$  добавляется метка  $!_i$  с индексом  $i$ , бóльшим всех имеющихся
- ▶ При откате с удалением вершины  $v$ , содержащей запрос вида  $?! , B_1, \dots, B_k$ , определяется метка  $!_i$  с наибольшим индексом  $i$ , и откат продолжается до вершины стека без этой метки

# Оператор отсечения (!) и стековые вычисления

## Пример

1 :  $\text{elem}(X, X.L) \leftarrow !;$

2 :  $\text{elem}(X, Y.L) \leftarrow \text{elem}(X, L); \quad ?\text{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$

Начинаем вычисление:

$? \text{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	$\varepsilon$	1
---	-------------	---------------	---

Правило 1 применить нельзя:

$? \text{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	$\varepsilon$	2
---	-------------	---------------	---

Применяем правило 2:

$? \text{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	$\varepsilon$	2
$? \text{elem}(\mathbf{0}, \mathbf{0.1.0.nil})$	$\emptyset$	$\varepsilon$	1

Применяем правило 1:

$? \text{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	$\varepsilon$	2	
$? \text{elem}(\mathbf{0}, \mathbf{0.1.0.nil})$	$\emptyset$	$\varepsilon$	1	$!_1$
$?!$	$\emptyset$	$\varepsilon$	1	$!_1$

# Оператор отсечения (!) и стековые вычисления

## Пример

1 :  $\text{elem}(X, X.L) \leftarrow !;$

2 :  $\text{elem}(X, Y.L) \leftarrow \text{elem}(X, L); \quad ?\text{elem}(0, 1.0.1.0.\text{nil})$

Выполняем !:

?elem(0, 1.0.1.0.nil)	$\emptyset$	$\varepsilon$	2	
?elem(0, 0.1.0.nil)	$\emptyset$	$\varepsilon$	1	! <sub>1</sub>
?!	$\emptyset$	$\varepsilon$	1	! <sub>1</sub>
□	$\emptyset$	$\varepsilon$	1	

Выдаём ответ  $\varepsilon$  («да»)

Откат:

?elem(0, 1.0.1.0.nil)	$\emptyset$	$\varepsilon$	2	
?elem(0, 0.1.0.nil)	$\emptyset$	$\varepsilon$	1	! <sub>1</sub>
?!	$\emptyset$	$\varepsilon$	2	! <sub>1</sub>

Левая подцель — !, продолжаем откат, пока не устраним метку !<sub>1</sub>:

?elem(0, 1.0.1.0.nil)	$\emptyset$	$\varepsilon$	3	
-----------------------	-------------	---------------	---	--

Правила кончились, откат

Стек пуст, обход завершён

# Оператор отсечения (!) и деревья вычислений

Эффект отсечения можно достаточно наглядно представить и в терминах **деревя вычислений**:

- ▶ При обходе дуги  $Q_1 \xrightarrow{\mathcal{R}} Q_2$ , такой что в  $\mathcal{R}$  содержится **!**, вершина  $Q_1$  (начало дуги) помечается меткой  $[i$  с уникальным индексом  $i$ 
  - ▶ Это место первой записи метки отсечения в стековом вычислении
- ▶ Когда знак **!**, появившийся при обходе дуги, упомянутой выше, становится левой подцелью, этот запрос помечается меткой  $]i$  для того же индекса  $i$ 
  - ▶ Это место последней записи метки отсечения в стековом вычислении
- ▶ При возврате в вершину с меткой  $]i$  выполняется перенаправление в родителя вершины с меткой  $[i$  (если родителя нет, то обход завершён)
  - ▶ То есть **отсекаются** все ветви дерева вычислений, которые вырастали бы в дальнейшем обходе из вершин от  $[i$  до  $]i$  включительно

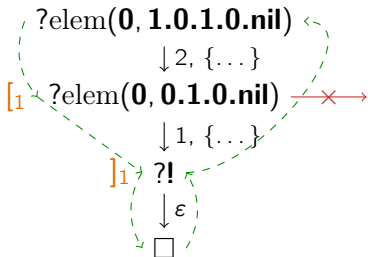


# Оператор отсечения (!) и деревья вычислений

## Пример

1 :  $\text{elem}(X, X.L) \leftarrow !;$

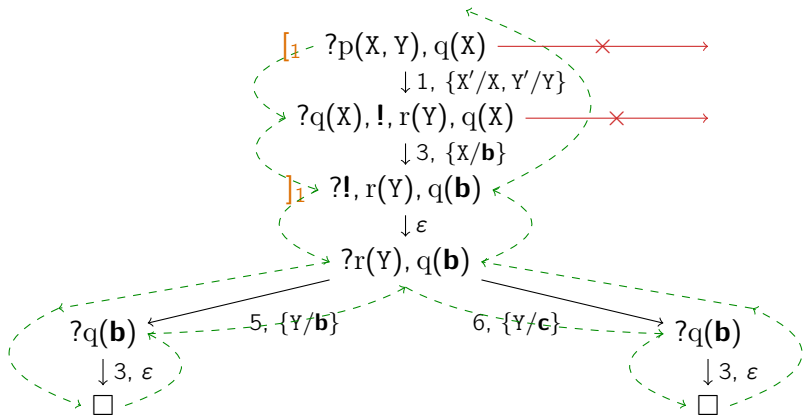
2 :  $\text{elem}(X, Y.L) \leftarrow \text{elem}(X, L); \quad ?\text{elem}(0, 1.0.1.0.\text{nil})$



# Оператор отсечения (!) и деревья вычислений

## Другой пример

$1 : p(X, Y) \leftarrow q(X), !, r(Y);$        $2 : p(X, X) \leftarrow r(X);$   
 $3 : q(\mathbf{b});$        $4 : q(\mathbf{c});$        $5 : r(\mathbf{b});$        $6 : r(\mathbf{c});$   
 $?p(X, Y), q(X)$



Ответ:  $\{X/\mathbf{b}, Y/\mathbf{b}\}$

Ответ:  $\{X/\mathbf{b}, Y/\mathbf{c}\}$

# Оператор отсечения (!) и деревья вычислений

Правило вида  $A \leftarrow B_1, \dots, B_k, !, C_1, \dots, C_m$  можно трактовать так: чтобы решить задачу  $A$ , следует проверить, верны ли условия  $B_1, \dots, B_k$  (найти первое попавшееся совокупное решение подзадач  $B_1, \dots, B_k$ , если оно есть), и

- ▶ если верны (решение найдено), то найти все решения подзадач  $C_1, \dots, C_m$  и не решать  $A$  другими способами,
- ▶ а если нет, то решить  $A$  другими способами

Отталкиваясь от этой трактовки, можно использовать **!** для моделирования конструкций императивной парадигмы:

- ▶ **Ветвление**     $A : \text{if } B \text{ then } C \text{ else } D \text{ fi}$   
                           $A \leftarrow B, !, C;$   
                           $A \leftarrow D;$

- ▶ **Цикл**         $A : \text{while } B \text{ do } C \text{ od}$   
                           $A \leftarrow B, !, C, A;$   
                           $A;$

# Оператор отсечения (!) и деревья вычислений

Следует иметь в виду, что для ХЛП с оператором отсечения перестаёт быть справедливой **теорема о полноте**, даже если дерево вычислений конечно: не для всякого правильного ответа можно вычислить какое-либо обобщение

**Например**, для программы

$$\begin{aligned} 1 &: \text{elem}(X, X.L) \leftarrow !; \\ 2 &: \text{elem}(X, Y.L) \leftarrow \text{elem}(X, L); \end{aligned}$$

и запроса

$$?\text{elem}(X, \mathbf{0.1.nil})$$

правильными ответами являются подстановки

$$\{X/\mathbf{0}\} \quad \text{и} \quad \{X/\mathbf{1}\},$$

а вычислен будет только ответ

$$\{X/\mathbf{0}\}$$

Поэтому оператор **!** следует использовать осторожно и только с хорошим пониманием того, какие ветви дерева вычислений будут отсечены и как это в целом повлияет на вычисление ответов