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

$$1 : \mathrm{elem}(\mathtt{X}, \mathtt{X.L}); \qquad 2 : \mathrm{elem}(\mathtt{X}, \mathtt{Y.L}) \leftarrow \mathrm{elem}(\mathtt{X}, \mathtt{L});$$

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

$$?\mathrm{elem}(\mathbf{0}, \mathbf{0.1.0.0.nil}) \xrightarrow{1,\ \{\mathtt{X}'/\mathbf{0}, \mathtt{L}'/\mathbf{1.0.0.nil}\}} \square \quad \text{Ответ: да } (\varepsilon)$$

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

$$?\mathrm{elem}(\mathbf{0}, \mathbf{1.0.0.nil})$$

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

$$?\mathrm{elem}(\mathbf{0}, \mathbf{0.0.nil}) \xrightarrow{1,\ \{\mathtt{X}'/\mathbf{0}, \mathtt{L}'/\mathbf{0.nil}\}} \square \quad \text{Ответ: да } (\varepsilon)$$

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

$$?\mathrm{elem}(\mathbf{0}, \mathbf{0.nil}) \xrightarrow{1,\ \{\mathtt{X}'/\mathbf{0}, \mathtt{L}'/\mathbf{0.nil}\}} \square \quad \text{Ответ: да } (\varepsilon)$$

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

$$?\mathrm{elem}(\mathbf{0}, \mathbf{nil})$$

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 вычисления

В декларативной семантике ! — это тавтология, формула $\mathbb{t}$

Как встроенный предикат ! всегда выполняется, и всегда с унификатором ε

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

Оператор отсечения (!) и стековые вычисления

Пример

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

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

Начинаем вычисление:

$?\mathrm{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	ε	1

Правило 1 применить нельзя:

$?\mathrm{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	ε	2

Применяем правило 2:

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

Применяем правило 1:

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

Оператор отсечения (!) и стековые вычисления

Пример

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

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

Выполняем **!**:

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

Выдаём ответ ε («да»)

Откат:

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

Левая подцель — **!**, продолжаем откат, пока не устраним метку $\mathbf{!}_1$:

$?\mathrm{elem}(\mathbf{0}, \mathbf{1.0.1.0.nil})$	$\emptyset$	ε	3	

Правила кончились, откат

Стек пуст, обход завершён

Оператор отсечения (!) и деревья вычислений

Эффект отсечения можно достаточно наглядно представить и в терминах дерева вычислений:

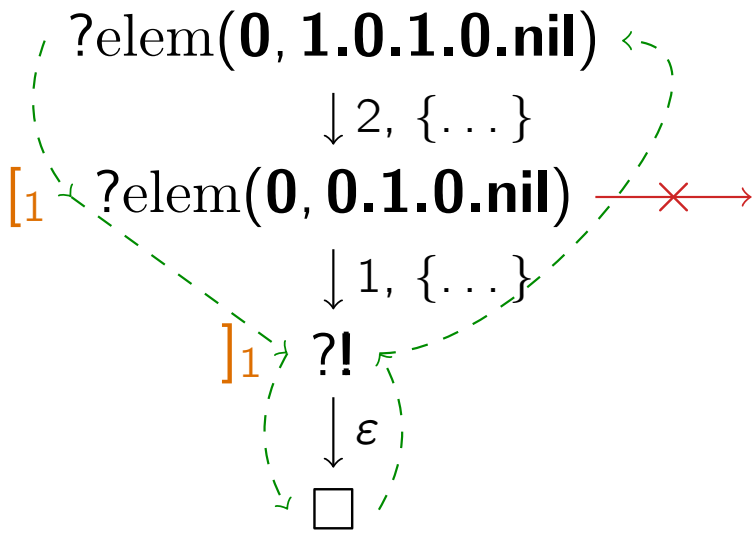
- При обходе дугй $\mathcal{Q}_1 \xrightarrow{\mathcal{R}} \mathcal{Q}_2$, такой что в $\mathcal{R}$ содержится **!**, вершина $\mathcal{Q}_1$ (начало дугй) помечается меткой $[_i$ с уникальным индексом i
  - Это место первой записи метки отсечения в стековом вычислении
- Когда знак **!**, появившийся 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

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

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



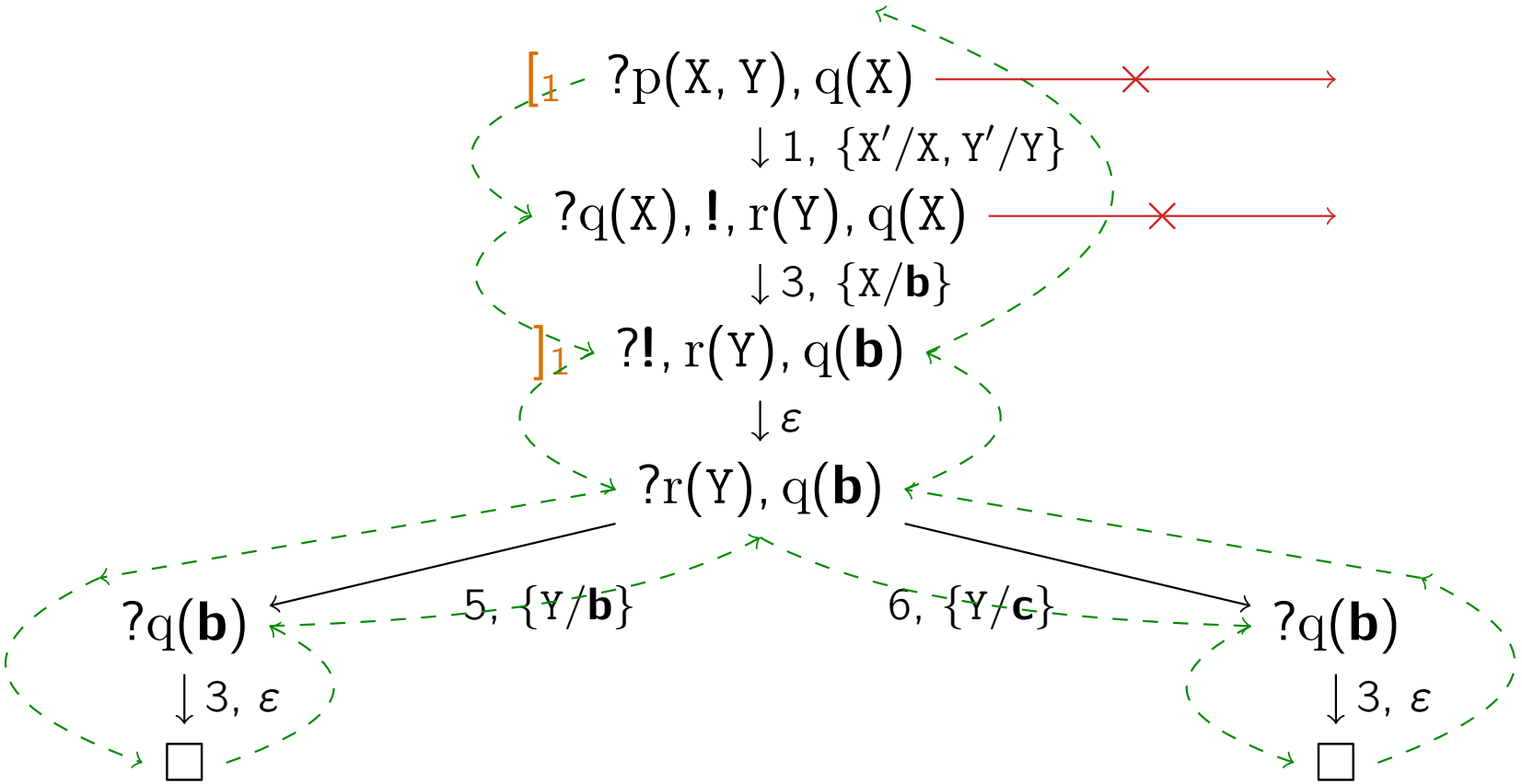
Оператор отсечения (!) и деревья вычислений

Другой пример

$1 : \mathrm{p}(\mathtt{X}, \mathtt{Y}) \leftarrow \mathrm{q}(\mathtt{X}), \mathbf{!}, \mathrm{r}(\mathtt{Y});$ $\quad 2 : \mathrm{p}(\mathtt{X}, \mathtt{X}) \leftarrow \mathrm{r}(\mathtt{X});$

$3 : \mathrm{q}(\mathbf{b});$ $\quad 4 : \mathrm{q}(\mathbf{c});$ $\quad 5 : \mathrm{r}(\mathbf{b});$ $\quad 6 : \mathrm{r}(\mathbf{c});$

$?\mathrm{p}(\mathtt{X}, \mathtt{Y}), \mathrm{q}(\mathtt{X})$



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

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

Оператор отсечения (!) и деревья вычислений

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

- если верны (решение найдено), то найти все решения подзадач $C_1, \ldots, C_m$ 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 : **if** B **then** C **else** D **fi**
  $A \leftarrow B, \mathbf{!}, C;$
  $A \leftarrow D;$
- Цикл A : **while** B **do** C **od**
  $A \leftarrow B, \mathbf{!}, C, A;$
  $A;$

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

$$1 : \mathrm{elem}(\mathtt{X}, \mathtt{X.L}) \leftarrow \mathbf{!};$$
$$2 : \mathrm{elem}(\mathtt{X}, \mathtt{Y.L}) \leftarrow \mathrm{elem}(\mathtt{X}, \mathtt{L});$$

и запроса

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

правильными ответами являются подстановки

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

а вычислен будет только ответ

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

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