Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 12.

Хорновские логические программы: синтаксис. Декларативная семантика логических программ. Операционная семантика логических программ: SLD–резолютивные вычисления.

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

ПАРАДИГМЫ ПРОГРАММИРОВАНИЯ

Императивное программирование : программа — это автомат, описывающий последовательности операторов (команд).
Математическая модель: машины Тьюринга–Поста.
Языки: **Assembler, Pascal, C, Java**.

Функциональное программирование : программа — это система уравнений, описывающая вычисляемую функцию.
Математическая модель: λ-исчисление Черча–Клини, уравнения Эрбрана–Геделя.
Языки: **Lisp, ML, Haskell**.

Логическое программирование : программа — это множество формул, описывающих условия решаемой задачи.
Математическая модель: логические исчисления.
Языки: **Prolog, Godel**.

ПАРАДИГМА ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

ОПИСАНИЕ ЗАДАЧИ

Декларативная семантика

ПРОГРАММА

Операционная семантика

ОПИСАНИЕ ЗАДАЧИ = ПРОГРАММА

Декларативная семантика $\Longleftrightarrow$ **Операционная семантика**

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Синтаксис логических программ

Пусть $\sigma = \langle Const, Func, Pred \rangle$ — некоторая сигнатура, в которой определяются термы и атомы.

«заголовок» ::= «атом»

«тело» ::= «атом» | «тело», «атом»

«правило» ::= «заголовок» $\leftarrow$ «тело»;

«факт» ::= «заголовок»;

«утверждение» ::= «правило» | «факт»

«программа» ::= «пусто» | «утверждение» «программа»

«запрос» ::= $\square$ | ? «тело»

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Примеры

Правило: $\underbrace{L(\textit{паша}, Y)}_{\text{заголовок}} \leftarrow \underbrace{L(Y, X),\ L(\textit{паша}, X)}_{\text{тело}};$

Факт: $L(\textit{даша}, \textit{саша})$;

Запрос (целевое утверждение):

$$? \ \underbrace{\text{Умный}(X)}_{\text{подцель}}, \ \underbrace{\text{Добрый}(X)}_{\text{подцель}}, \ \underbrace{\text{Красивый}(X)}_{\text{подцель}}, \ \underbrace{\text{Любит}(X, \textit{меня})}_{\text{подцель}}$$

Здесь X — целевая переменная .

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Пример логической программы

$$
\begin{aligned}
\mathcal{P}: \quad & \mathit{elem}(X, X \centerdot L); \\
& \mathit{elem}(X, Y \centerdot L) \leftarrow \mathit{elem}(X, L); \\
& \mathit{concat}(\mathbf{nil}, Y, Y); \\
& \mathit{concat}(U \centerdot X, Y, U \centerdot Z) \leftarrow \mathit{concat}(X, Y, Z); \\
& \mathit{common}(X, Y) \leftarrow \mathit{elem}(U, X), \mathit{elem}(U, Y);
\end{aligned}
$$

и запроса к ней

$$
? \ \mathit{concat}(L1, L2, \text{п} \centerdot \text{р} \centerdot \text{о} \centerdot \text{г} \centerdot \text{р} \centerdot \text{а} \centerdot \text{м} \centerdot \text{м} \centerdot \text{а} \centerdot \mathbf{nil}), \mathit{common}(L1, L2);
$$

Какой ответ мы ожидаем получить на этот запрос?

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Пример логической программы

```
simp_path(X,X,Vert,Arc,nil) ← elem(X,Vert,U);

simp_path(X,Y,V,A,Path)     ← elem(X,V,U1),elem(Y,U1,U2),
                              find_path(X,Y,U2,A,P);

find_path(X,Y,V,A,(X.Y.nil).nil) ← elem(X.Y.nil,A,A1);

find_path(X,Y,V,A,(X.Z.nil).Path) ← elem(Z,V,V1),
                                    elem(X.Z.nil,A,A1),
                                    find_path(Z,Y,V1,A1,Path);
elem(X,X.L1,L1);

elem(X,Y.L1,Y.L1) ← elem(X,L1,L2);
```

и запроса к ней

```
?simp_path(4,2,1.2.3.4.nil,(1.2.nil).(2.3.nil).(2.4.nil).(3.1.nil).nil,X)
```

Что же вычислит программа в ответ на этот запрос?

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Как нужно понимать логические программы?

Главная особенность логического программирования — полисемантичность: одна и та же логическая программа имеет две равноправные семантики, два смысла.
Человек–программист и компьютер–вычислитель имеют две разные точки зрения на программу.

Программисту важно понимать, **ЧТО** вычисляет программа. Такое понимание программы называется декларативной семантикой программы.

Компьютеру важно «знать», **КАК** проводить вычисление программы. Такое понимание программы называется операционной семантикой программы.

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Как нужно понимать логические программы?

Декларативная семантика	Операционная семантика
Правило $A_0 \leftarrow A_1, A_2, \ldots, A_n$;	
Если выполнены условия $A_1, A_2, \ldots, A_n$, то справедливо и утверждение A_0.	Чтобы решить задачу A_0, достаточно решить задачи $A_1, A_2, \ldots, A_n$.
Факт A_0;	
Утверждение A_0 считается верным.	Задача A_0 объявляется решенной.
Запрос $?C_1, C_2, \ldots, C_m$	
При каких значениях целевых переменных будут верны все отношения $C_1, C_2, \ldots, C_m$?	Решить список задач $C_1, C_2, \ldots, C_m$.

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Пример истолкования логической программы

$$\begin{aligned} \mathcal{P}: \quad & elem(X, X \centerdot L); \\ & elem(X, Y \centerdot L) \quad \leftarrow \quad elem(X, L); \end{aligned}$$

Декларативная семантика	Операционная семантика
1. Всякий предмет X входит в состав того списка, заголовком которого он является	1. Считается решенной задача поиска предмета X в любом списке, содержащем X в качестве заголовка.
2. Если предмет X содержится в хвосте списка, то X содержится и в самом списке.	2. Чтобы обнаружить предмет X в списке, достаточно найти его в хвосте этого списка.

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Терминология

Пусть $\mathcal{P}$ — логическая программа, D — программное утверждение, а θ — подстановка. Тогда

- $D\theta$ — пример программного утверждения D,
- если θ — переименование, то $D\theta$ — вариант программного утверждения D,
- если $Var_{D\theta} = \emptyset$, то $D\theta$ — основной пример программного утверждения D,
- $[D]$ — множество всех основных примеров программного утверждения D,
- $[\mathcal{P}]$ — множество всех основных примеров всех утверждений программы $\mathcal{P}$.

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Терминология

Пусть $G = ?C_1, C_2, \ldots, C_m$ — запрос. Тогда

- атомы $C_1, C_2, \ldots, C_m$ называются подцелями запроса G,
- переменные множества $\bigcup_{i=1}^{m} Var_{C_i}$ называются целевыми переменными ,
- запрос $\square$ называется пустым запросом ,
- запросы будем также называть целевыми утверждениями .

Для удобства обозначения условимся в дальнейшем факты A; рассматривать как правила $A \leftarrow$; с заголовком A и пустым телом.

$elem(X, X . L);$ $\qquad$ $elem(X, X . L) \leftarrow;$

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Примеры

Пусть $D = \ elem(X, Y \centerdot Z) \ \leftarrow \ elem(X, Z)$; — программное утверждение. Тогда

$D' = D\{X/Y, Z/\mathbf{nil}\} = \ elem(Y, Y \centerdot \mathbf{nil}) \ \leftarrow \ elem(Y, \mathbf{nil})$;
пример программного утверждения D,

$D'' = \ elem(X', Y' \centerdot Z') \ \leftarrow \ elem(X', Z')$;
вариант программного утверждения D,

$D''' = D\{X/1, Y/2, Z/\mathbf{nil}\} = \ elem(1, 2 \centerdot \mathbf{nil}) \ \leftarrow \ elem(1, \mathbf{nil})$;
основной пример программного утверждения D,

ДЕКЛАРАТИВНАЯ СЕМАНТИКА

Более строгое описание семантик требует привлечения аппарата математической логики.

Логические программы и логические формулы

Каждому утверждению логической программы сопоставим логическую формулу:

Правило: $D' = A_0 \leftarrow A_1, A_2, \ldots, A_n$;

$$D' = \forall X_1 \ldots \forall X_k (A_1 \& A_2 \& \ldots \& A_n \rightarrow A_0), \text{ где } \{X_1, \ldots, X_k\} = \bigcup_{i=0}^{n} Var_{A_i}$$

Факт: $D'' = A$;

$$D'' = \forall X_1 \ldots \forall X_k A, \text{ где } \{X_1, \ldots, X_k\} = Var_A$$

Запрос: $G = ?\ C_1, C_2, \ldots, C_m$

$$G = C_1 \& C_2 \& \ldots \& C_m$$

ДЕКЛАРАТИВНАЯ СЕМАНТИКА

С точки зрения декларативной семантики,

- программные утверждения D и запросы G — это логические формулы,
- программа $\mathcal{P}$ — это множество формул (база знаний),
- а правильный ответ на запрос — это такие значения переменных (подстановка), при которой запрос оказывается логическим следствием базы знаний.

ДЕКЛАРАТИВНАЯ СЕМАНТИКА

Определение (правильного ответа)

Пусть $\mathcal{P}$ — логическая программа, G — запрос к $\mathcal{P}$ с множеством целевых переменных $Y_1, \ldots, Y_k$.

Тогда всякая подстановка $\theta = \{Y_1/t_1, \ldots, Y_k/t_k\}$ называется ответом на запрос G к программе $\mathcal{P}$.

Ответ $\theta = \{Y_1/t_1, \ldots, Y_k/t_k\}$ называется правильным ответом на запрос G к программе $\mathcal{P}$, если

$$\mathcal{P} \models \forall Z_1 \ldots \forall Z_N G\theta, \qquad \text{где } \{Z_1, \ldots, Z_N\} = \bigcup_{i=1}^{k} Var_{t_i}.$$

ХОРНОВСКИЕ ЛОГИЧЕСКИЕ ПРОГРАММЫ

Примеры правильных ответов

Правильными ответами на запрос G : ? $elem(X, \text{с}\blacksquare\,\text{т}\blacksquare\,\text{о}\blacksquare\,\text{л}\blacksquare\,\textbf{nil})$; обращенный к логической программе

$$\begin{aligned}\mathcal{P}: \quad & elem(X, X\blacksquare\,L);\\ & elem(X, Y\blacksquare\,L) \leftarrow elem(X, L);\end{aligned}$$

являются четыре подстановки

$$\theta_1 = \{X/\text{с}\},\ \theta_2 = \{X/\text{т}\}, \theta_3 = \{X/\text{о}\}, \theta_4 = \{X/\text{л}\},$$

поскольку для любой из этих подстановок верно соотношение

$$\{\forall X \forall Y \forall L(elem(X, L) \rightarrow elem(X, Y\blacksquare\,L)),\ \forall X \forall L\ elem(X, X\blacksquare\,L)\} \models elem(X, \text{с}\blacksquare\,\text{т}\blacksquare\,\text{о}\blacksquare\,\text{л}\blacksquare\,\textbf{nil})\theta_i\ .$$

Но как искать правильные ответы?

ОПЕРАЦИОННАЯ СЕМАНТИКА ЛОГИЧЕСКИХ ПРОГРАММ

Концепция операционной семантики

Под операционной семантикой понимают правила построения **вычислений программы**. Операционная семантика описывает, КАК достигается результат работы программы.

Ожидаемый результат работы логической программы — это **правильный ответ** на запрос к программе. Значит, операционная семантика должна описывать метод вычисления правильных ответов.

Таким методом вычисления может быть разновидность **метода резолюций**, учитывающая особенности устройства программных утверждений.

ОПЕРАЦИОННАЯ СЕМАНТИКА

Логически предпосылки операционной семантики

Запрос $G(Y_1, \ldots Y_m) =? C_1, C_2, \ldots, C_m$ к логической программе $\mathcal{P} = \{D_1, \ldots, D_N\}$ порождает задачу о логическом следствии:

$$\{D_1, \ldots, D_N\} \models \exists Y_1 \ldots \exists Y_k (C_1 \& C_2 \& \ldots \& C_m),$$

которая равносильна задаче об общезначимости

$$\models D_1 \& \ldots \& D_N \ \rightarrow \ \exists Y_1 \ldots \exists Y_k (C_1 \& C_2 \& \ldots \& C_m),$$

которая равносильна задаче о противоречивости формулы

$$\neg (D_1 \& \ldots \& D_N \ \rightarrow \ \exists Y_1 \ldots \exists Y_k (C_1 \& C_2 \& \ldots \& C_m),$$

равносильной формуле

$$D_1 \& \ldots \& D_N \ \& \ \forall Y_1 \ldots \forall Y_k (\neg C_1 \vee \neg C_2 \vee \cdots \vee \neg C_m),$$

ОПЕРАЦИОННАЯ СЕМАНТИКА

Логически предпосылки операционной семантики

Полученную формулу

$$D_1 \& \ldots \& D_N \ \& \ \forall Y_1 \ldots \forall Y_k (\neg C_1 \vee \neg C_2 \vee \cdots \vee \neg C_m),$$

можно рассматривать как систему дизъюнктов

$$S_{\mathcal{P},G} \ = \ \{D_1, \ldots, D_N, \neg C_1 \vee \neg C_2 \vee \cdots \vee \neg C_m\},$$

и доказывать ее противоречивость методом резолюций.

ОПЕРАЦИОННАЯ СЕМАНТИКА

Логические программы и хорновские дизъюнкты

Каждому утверждению логической программы сопоставим хорновский дизъюнкт:

Правило: $D' = A_0 \leftarrow A_1, A_2, \ldots, A_n$

$$D' = A_0 \vee \neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n$$

Факт: $D'' = A$

$$D'' = A$$

Запрос: $G = ?\ C_1, C_2, \ldots, C_m$

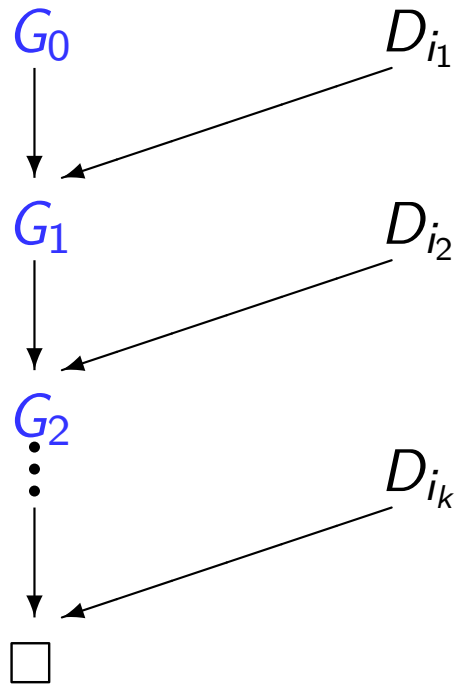
$$G = \neg C_1 \vee \neg C_2 \vee \cdots \vee \neg C_m$$

Как это принято у дизъюнктов, предполагается, что все переменные связаны кванторами $\forall$.

ОПЕРАЦИОННАЯ СЕМАНТИКА

Логические программы и хорновские дизъюнкты

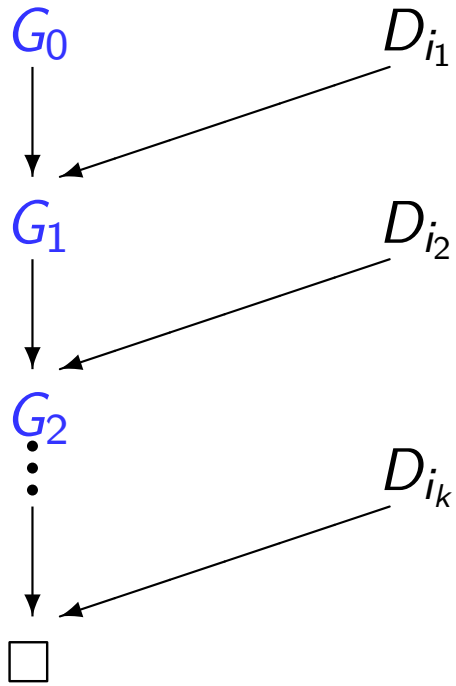
Мы будем применять специальную стратегию построения резолютивного вывода:



ОПЕРАЦИОННАЯ СЕМАНТИКА

Логические программы и хорновские дизъюнкты

Мы будем применять специальную стратегию построения резолютивного вывода:



L inear resolution with S election function for D efinite clauses

SLD-резолюция (Р. Ковальски)

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

Пусть

- $G = ?\ C_1, \ldots, C_i, \ldots, C_m$ — целевое утверждение, в котором выделена подцель C_i,
- $D' = A'_0 \leftarrow \mathbf{A'_1}, \mathbf{A'_2}, \ldots, \mathbf{A'_n}$ — **вариант** некоторого программного утверждения, в котором $Var_G \cap Var_{D'} = \emptyset$,
- $\theta \in HOY(C_i, A'_0)$ — наиб. общ. унификатор подцели C_i и заголовка программного утверждения A'_0.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

Пусть

- $G = ?\ C_1, \ldots, C_i, \ldots, C_m$ — целевое утверждение, в котором выделена подцель C_i,
- $D' = A'_0 \leftarrow \mathbf{A'_1}, \mathbf{A'_2}, \ldots, \mathbf{A'_n}$ — **вариант** некоторого программного утверждения, в котором $Var_G \cap Var_{D'} = \emptyset$,
- $\theta \in HOY(C_i, A'_0)$ — наиб. общ. унификатор подцели C_i и заголовка программного утверждения A'_0.

Тогда запрос

$$G' = ?(C_1, \ldots, C_{i-1}, \mathbf{A'_1}, \mathbf{A'_2}, \ldots, \mathbf{A'_n}, C_{i+1}, \ldots, C_m)\theta$$

называется SLD-резольвентой программного утверждения D' и запроса G с выделенной подцелью C_i и унификатором θ.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$G = ?\ C_1, \ldots, C_i, \ldots, C_m$

КОММЕНТАРИИ.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$G = ?\ C_1, \ldots, C_i, \ldots, C_m$

КОММЕНТАРИИ.

Выделяем подцель в запросе.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$$G = ?\ C_1, \ldots, C_i, \ldots, C_m$$

$$D = \ A_0 \leftarrow A_1, A_2, \ldots, A_n;$$

КОММЕНТАРИИ.

Выбираем программное утверждение.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$$G = ?\ C_1, \ldots, C_i, \ldots, C_m$$

$$D' = A'_0 \leftarrow A'_1, A'_2, \ldots, A'_n;$$

КОММЕНТАРИИ.

Переименовываем переменные в выбранном утверждении, так чтобы $Var_{D'} \cap Var_G = \emptyset$.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$$G = ?\ C_1, \ldots, C_i, \ldots, C_m$$

$$D' = A'_0 \leftarrow A'_1, A'_2, \ldots, A'_n;$$

$$\theta = НОУ(C_i, A'_0)$$

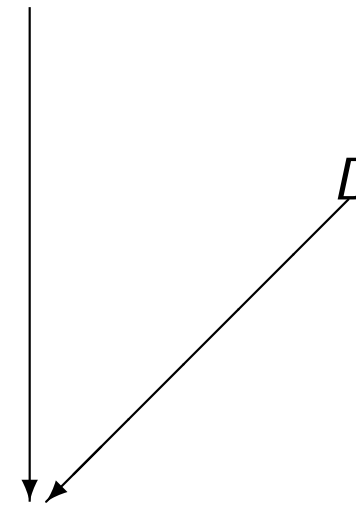
КОММЕНТАРИИ.

Вычисляем Наиболее Общий Унификатор.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$G = ?\ C_1, \ldots, C_i, \ldots, C_m$



$D' = A'_0 \leftarrow A'_1, A'_2, \ldots, A'_n;$

$\theta = НОУ(C_i, A'_0)$

$G' = ?\ (C_1, \ldots, C_{i-1}, A'_1, A'_2, \ldots, A'_n, C_{i+1}, \ldots, C_m)\theta$

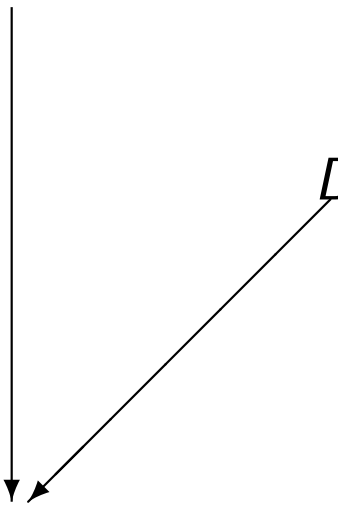
КОММЕНТАРИИ.

Строим SLD-резольвенту

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолюции)

$$G = \neg C_1 \vee \cdots \vee \neg C_i \vee \cdots \vee \neg C_m$$



$$D' = A'_0 \vee \neg A'_1 \vee \neg A'_2 \vee \cdots \vee A'_n;$$

$$\theta = НОУ(C_i, A'_0)$$

$$G' = (\neg C_1 \vee \cdots \vee \neg C_{i-1} \vee \neg A'_1 \vee \neg A'_2 \vee \cdots \vee \neg A'_n \vee \neg C_{i+1} \vee \cdots \vee \neg C_m)\theta$$

КОММЕНТАРИИ.

Действительно, это резольвента двух дизъюнктов

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$$G = ?\ P(X), R(X, f(Y)), R(Y, c)$$

КОММЕНТАРИИ.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$$G = ?\ P(X), R(X, f(Y)), R(Y, c)$$

КОММЕНТАРИИ.

Выделяем подцель в запросе.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$$G = ?\ P(X), R(X, f(Y)), R(Y, c)$$

$$D = \ R(Y, X) \leftarrow P(X), R(c, Y);$$

КОММЕНТАРИИ.

Выбираем программное утверждение.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$$G = ?\ P(X), R(X, f(Y)), R(Y, c)$$

$$D' = \ R(Y_1, X_1) \leftarrow P(X_1), R(c, Y_1);$$

КОММЕНТАРИИ.

Переименовываем переменные в выбранном утверждении, так чтобы $Var_{D'} \cap Var_G = \emptyset$.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$$G = ?\ P(X), R(X, f(Y)), R(Y, c)$$

$$D' = \ R(Y_1, X_1) \leftarrow P(X_1), R(c, Y_1);$$

$$\theta = НОУ(R(X, f(Y)), R(Y_1, X_1)) = \{Y_1/X, X_1/f(Y)\}$$

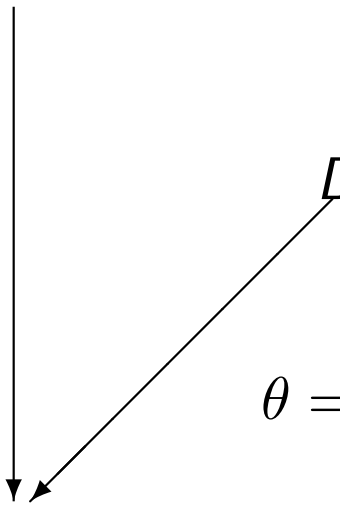
КОММЕНТАРИИ.

Вычисляем Наиболее Общий Унификатор.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$G = ?\ P(X), R(X, f(Y)), R(Y, c)$



$D' = R(Y_1, X_1) \leftarrow P(X_1), R(c, Y_1);$

$\theta = НОУ(R(X, f(Y)), R(Y_1, X_1)) = \{Y_1/X, X_1/f(Y)\}$

$G' = ?\ (P(X), \mathbf{P(X_1)}, \mathbf{R(c, Y_1)}, R(Y, c))\theta$

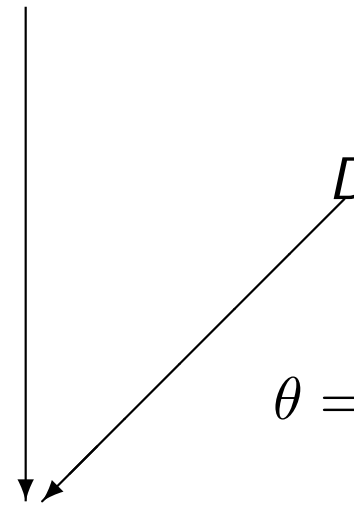
КОММЕНТАРИИ.

Строим SLD-резольвенту

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 1.

$G = ?\ P(X), R(X, f(Y)), R(Y, c)$



$D' = R(Y_1, X_1) \leftarrow P(X_1), R(c, Y_1);$

$\theta = НОУ(R(X, f(Y)), R(Y_1, X_1)) = \{Y_1/X, X_1/f(Y)\}$

$G' = ?\ P(X), P(f(Y)), R(c, X), R(Y, c)$

КОММЕНТАРИИ.

Вот она.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.

$G = ?\ R(X, X \,.\, \mathbf{nil})$

КОММЕНТАРИИ.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.

$G = ?\ R(X, X \,.\, \mathbf{nil})$

КОММЕНТАРИИ.

Выделяем подцель в запросе.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.

$G = ?\ R(X, X \,.\, \mathbf{nil})$

$$D = \ R(X \,.\, \mathbf{nil}, Y) \leftarrow;$$

КОММЕНТАРИИ.

Выбираем программное утверждение.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.

$G = ?\ R(X, X \,.\, \mathbf{nil})$

$$D' = \ R(X_1 \,.\, \mathbf{nil}, Y_1) \leftarrow;$$

КОММЕНТАРИИ.

Переименовываем переменные в выбранном утверждении, так чтобы $Var_{D'} \cap Var_G = \emptyset$.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.

$$G = ?\ R(X, X \,.\, \mathbf{nil})$$

$$D' = \ R(X_1 \,.\, \mathbf{nil}, Y_1) \leftarrow;$$

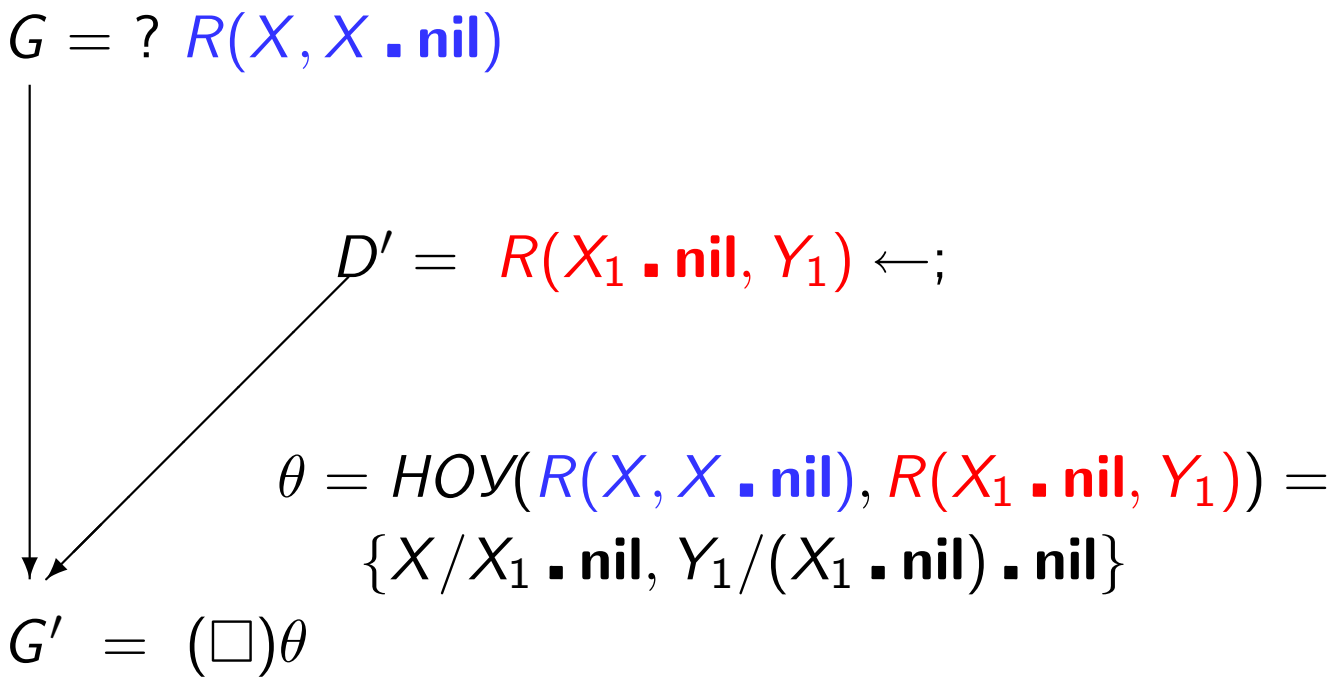
$$\theta = \mathit{НОУ}(R(X, X \,.\, \mathbf{nil}), R(X_1 \,.\, \mathbf{nil}, Y_1)) = \{X/X_1 \,.\, \mathbf{nil}, Y_1/(X_1 \,.\, \mathbf{nil}) \,.\, \mathbf{nil}\}$$

КОММЕНТАРИИ.

Вычисляем Наиболее Общий Унификатор.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.



КОММЕНТАРИИ.

Строим SLD-резольвенту

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 2.

$G = ?\ R(X, X \centerdot \mathbf{nil})$

$D' = R(X_1 \centerdot \mathbf{nil}, Y_1) \leftarrow;$

$\theta = НОУ(R(X, X \centerdot \mathbf{nil}), R(X_1 \centerdot \mathbf{nil}, Y_1)) =$
$\{X/X_1 \centerdot \mathbf{nil}, Y_1/(X_1 \centerdot \mathbf{nil}) \centerdot \mathbf{nil}\}$

$G' = \square$

КОММЕНТАРИИ.

Вот она.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$$G = ?\ R(X, X \,.\, \mathbf{nil}), P(c, Y)$$

КОММЕНТАРИИ.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$G = ?\ R(X, X \,.\, \mathbf{nil}), P(c, Y)$

КОММЕНТАРИИ.

Выделяем подцель в запросе.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$$G = ?\ R(X, X \centerdot \mathbf{nil}), P(c, Y)$$

$$D = \ R(X \centerdot \mathbf{nil}, X) \leftarrow;$$

КОММЕНТАРИИ.

Выбираем программное утверждение.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$$G = ?\ R(X, X \centerdot \mathbf{nil}), P(c, Y)$$

$$D' = \ R(X_1 \centerdot \mathbf{nil}, X_1) \leftarrow;$$

КОММЕНТАРИИ.

Переименовываем переменные в выбранном утверждении,

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$$G = ?\ R(X, X \centerdot \mathbf{nil}), P(c, Y)$$

$$D' = \ R(X_1 \centerdot \mathbf{nil}, X_1) \leftarrow;$$

$$НОУ(R(X, X \centerdot \mathbf{nil}), R(X_1 \centerdot \mathbf{nil}, X_1)) = \emptyset$$

КОММЕНТАРИИ.

Атомы неунифицируемы!

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$$G = ?\ R(X, X \centerdot \mathbf{nil}), P(c, Y)$$

$$D' = \ R(X_1 \centerdot \mathbf{nil}, X_1) \leftarrow;$$

$$НОУ(R(X, X \centerdot \mathbf{nil}), R(X_1 \centerdot \mathbf{nil}, X_1)) = \emptyset$$

КОММЕНТАРИИ.

Значит, SLD-резольвенту нельзя построить.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 3.

$$G = ?\ R(X, X \centerdot \mathbf{nil}), P(c, Y)$$

$$D' = \ R(X_1 \centerdot \mathbf{nil}, X_1) \leftarrow;$$

$$НОУ(R(X, X \centerdot \mathbf{nil}), R(X_1 \centerdot \mathbf{nil}, X_1)) = \emptyset$$

КОММЕНТАРИИ.

Нужно выделить другую подцель или

выбрать другое программное утверждение.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолютивного вычисления)

Пусть

- $G_0 = ?\ C_1, C_2, \ldots, C_m$ — целевое утверждение,
- $\mathcal{P} = \{D_1, D_2, \ldots, D_N\}$ — хорновская логическая программа.

Тогда (частичным) SLD-резолютивным вычислением , порожденным запросом G_0 к логической программе $\mathcal{P}$ называется последовательность троек (конечная или бесконечная)

$$(D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \ldots, (D_{j_n}, \theta_n, G_n), \ldots,$$

в которой для любого i, $i \geq 1$,

- $D_{j_i} \in \mathcal{P}$, $\theta_i \in Subst$, G_i — целевое утверждение (запрос);
- запрос G_i является SLD-резольвентой программного утверждения D_{j_i} и запроса G_{i-1} с унификатором θ_i.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолютивного вычисления)

Частичное SLD-резолютивное вычисление

$$comp = (D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \ldots, (D_{j_k}, \theta_n, G_n)$$

называется

- успешным вычислением (SLD-резолютивным опровержением), если $G_n = \square$;
- бесконечным вычислением , если *comp* — это **бесконечная** последовательность;
- тупиковым вычислением , если *comp* — это **конечная** последовательность, и при этом для выделенной подцели запроса G_n невозможно построить ни одной SLD-резольвенты.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Определение (SLD-резолютивного вычисления)

Пусть

- $G_0 = ?\ C_1, C_2, \ldots, C_m$ — целевое утверждение с целевыми переменными $Y_1, Y_2, \ldots, Y_k$,
- $\mathcal{P} = \{D_1, D_2, \ldots, D_N\}$ — хорновская логическая программа,
- $comp = (D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \ldots, (D_{j_n}, \theta_n, \square)$ — успешное SLD-резолютивное вычисление, порожденное запросом G к программе $\mathcal{P}$.

Тогда подстановка $\theta = (\theta_1\theta_2\ldots\theta_n)|_{Y_1, Y_2, \ldots, Y_k}$,

представляющая собой композицию всех вычисленных унификаторов $\theta_1,\ \theta_2, \ldots, \theta_n$, ограниченную целевыми переменными $Y_1, Y_2, \ldots, Y_k$,

называется вычисленным ответом на запрос G_0 к программе $\mathcal{P}$.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X/X_1,\ Y_1/a,\ L_1/b \centerdot c \centerdot \mathbf{nil}\}$

$G_1 = ?elem(X_1, b \centerdot c \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X/X_1,\ Y_1/a,\ L_1/b \centerdot c \centerdot \mathbf{nil}\}$

$G_1 = ?elem(X_1, b \centerdot c \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X/X_1,\ Y_1/a,\ L_1/b \centerdot c \centerdot \mathbf{nil}\}$

$G_1 = ?elem(X_1, b \centerdot c \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/c \centerdot \mathbf{nil}\}$

$G_2 = ?elem(X_2, c \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X/X_1,\ Y_1/a,\ L_1/b \centerdot c \centerdot \mathbf{nil}\}$

$G_1 = ?elem(X_1, b \centerdot c \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/c \centerdot \mathbf{nil}\}$

$G_2 = ?elem(X_2, c \centerdot \mathbf{nil})$

$elem(X_3, X_3 \centerdot \mathbf{nil})$;

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$G_0 = ?elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X/X_1,\ Y_1/a,\ L_1/b \centerdot c \centerdot \mathbf{nil}\}$

$G_1 = ?elem(X_1, b \centerdot c \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/c \centerdot \mathbf{nil}\}$

$G_2 = ?elem(X_2, c \centerdot \mathbf{nil})$

$elem(X_3, X_3 \centerdot \mathbf{nil})$;

$\theta_3 = \{X_2/c, X_3/c\}$

$G_3 = \square$

УСПЕХ!

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 4.

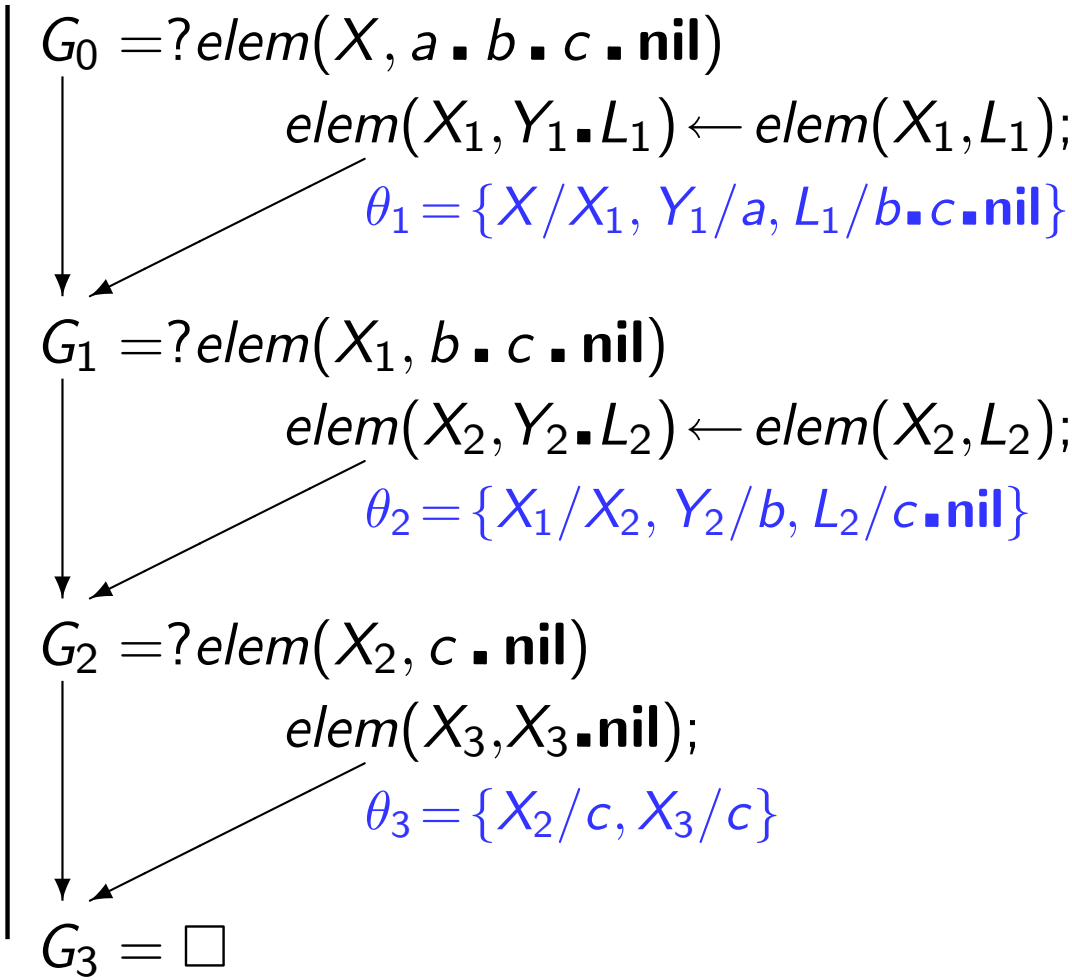
Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(X, a \centerdot b \centerdot c \centerdot \mathbf{nil})$



УСПЕХ!

Вычисленный ответ: $\theta = (\theta_1\theta_2\theta_3)|_X = \{X/c\}$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X_1/c, Y_1/a, L_1/b \centerdot \mathbf{nil}\}$

$G_1 = ?elem(c, b \centerdot \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X_1/c,\ Y_1/a,\ L_1/b \centerdot \mathbf{nil}\}$

$G_1 = ?elem(c, b \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X_1/c,\ Y_1/a,\ L_1/b \centerdot \mathbf{nil}\}$

$G_1 = ?elem(c, b \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/\mathbf{nil}\}$

$G_2 = ?elem(c, \mathbf{nil})$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X_1/c,\ Y_1/a,\ L_1/b \centerdot \mathbf{nil}\}$

$G_1 = ?elem(c, b \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/\mathbf{nil}\}$

$G_2 = ?elem(c, \mathbf{nil})$

$elem(X_3, X_3 \centerdot L_3)$;

Нет унификатора

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X_1/c,\ Y_1/a,\ L_1/b \centerdot \mathbf{nil}\}$

$G_1 = ?elem(c, b \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/\mathbf{nil}\}$

$G_2 = ?elem(c, \mathbf{nil})$

$elem(X_3, Y_3 \centerdot L_3) \leftarrow elem(X_3, L_3)$;

Нет унификатора

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 5.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(c, a \centerdot b \centerdot \mathbf{nil})$

$G_0 = ?elem(c, a \centerdot b \centerdot \mathbf{nil})$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

$\theta_1 = \{X_1/c,\ Y_1/a,\ L_1/b \centerdot \mathbf{nil}\}$

$G_1 = ?elem(c, b \centerdot \mathbf{nil})$

$elem(X_2, Y_2 \centerdot L_2) \leftarrow elem(X_2, L_2)$;

$\theta_2 = \{X_1/X_2,\ Y_2/b,\ L_2/\mathbf{nil}\}$

$G_2 = ?elem(c, \mathbf{nil})$

failure

Нет SLD-резольвенты

ТУПИК!

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$

$G_0 = ?elem(a, X)$

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$

$G_0 = ?elem(a, X)$

$elem(X_1, Y_1 \centerdot L_1) \leftarrow elem(X_1, L_1)$;

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

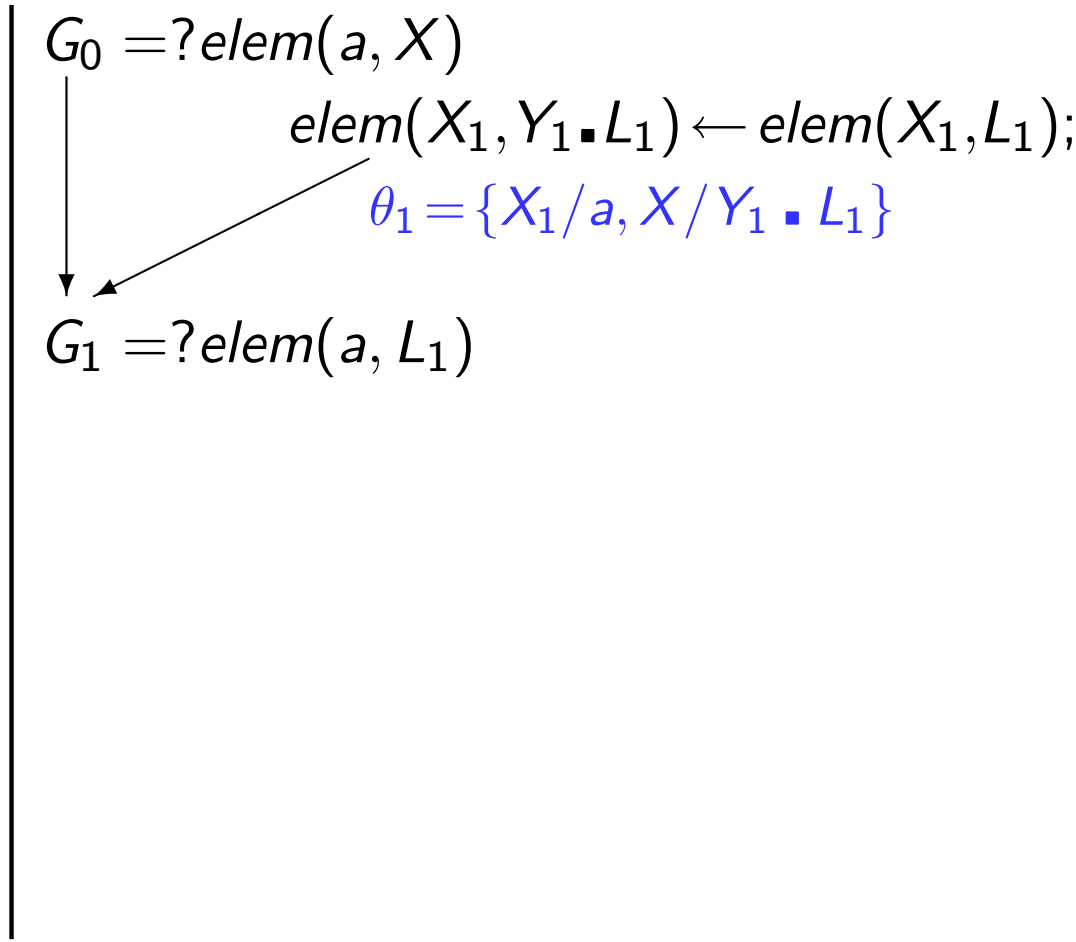
Логическая программа $\mathcal{P}$:

$elem(X, X \cdot L)$;

$elem(X, Y \cdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$



SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

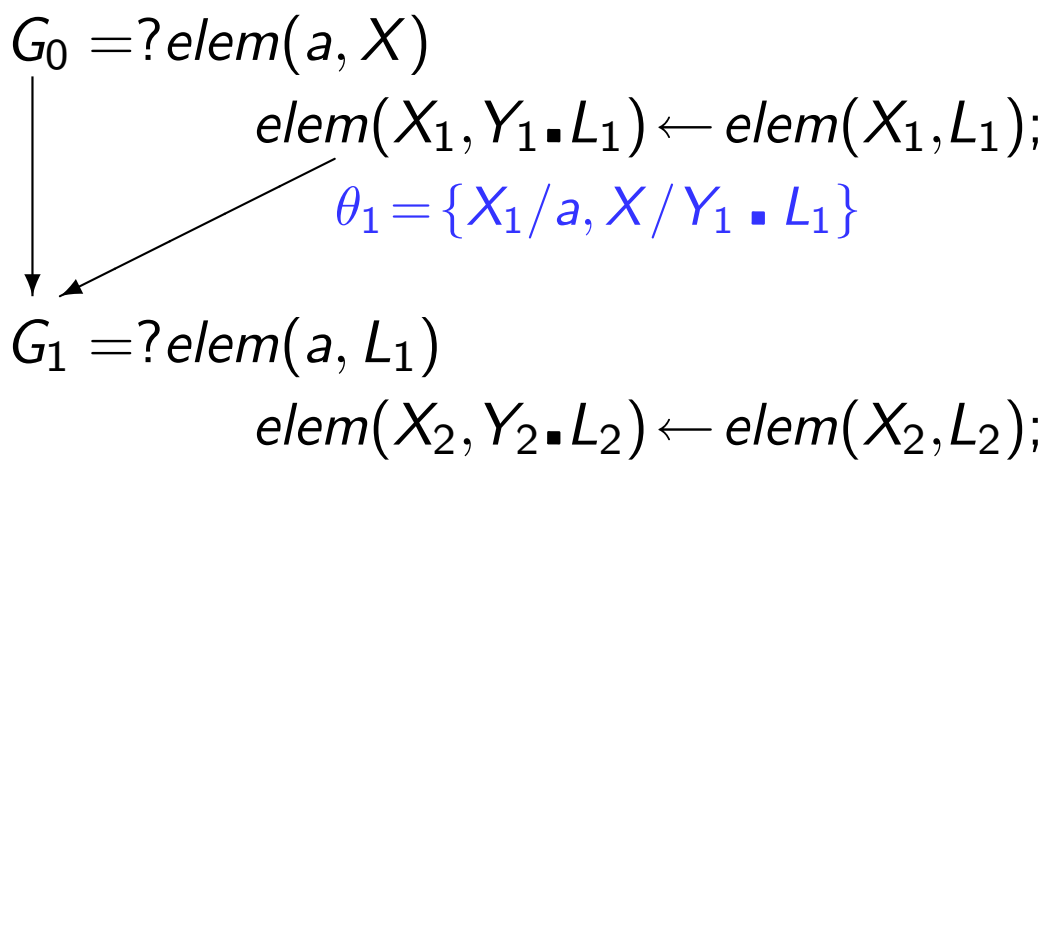
Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$



SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

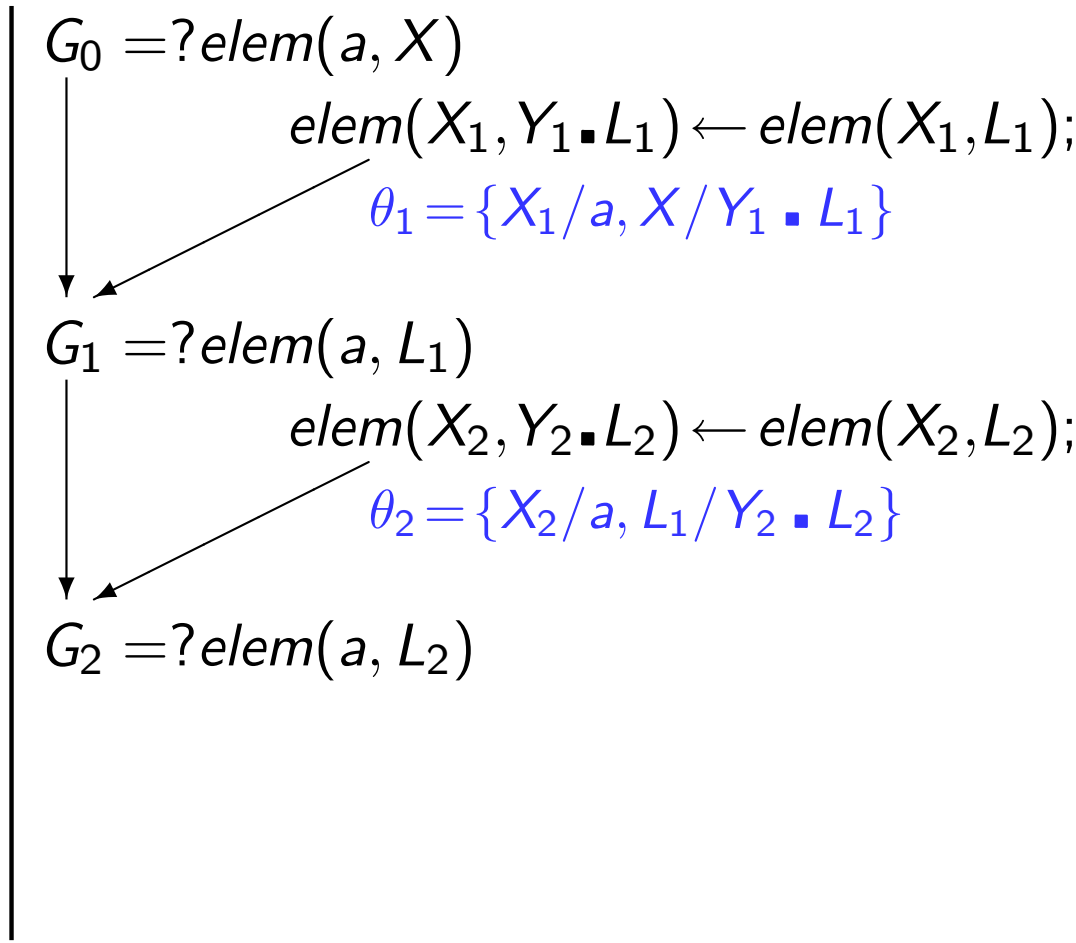
Логическая программа $\mathcal{P}$:

$elem(X, X \cdot L)$;

$elem(X, Y \cdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$



SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

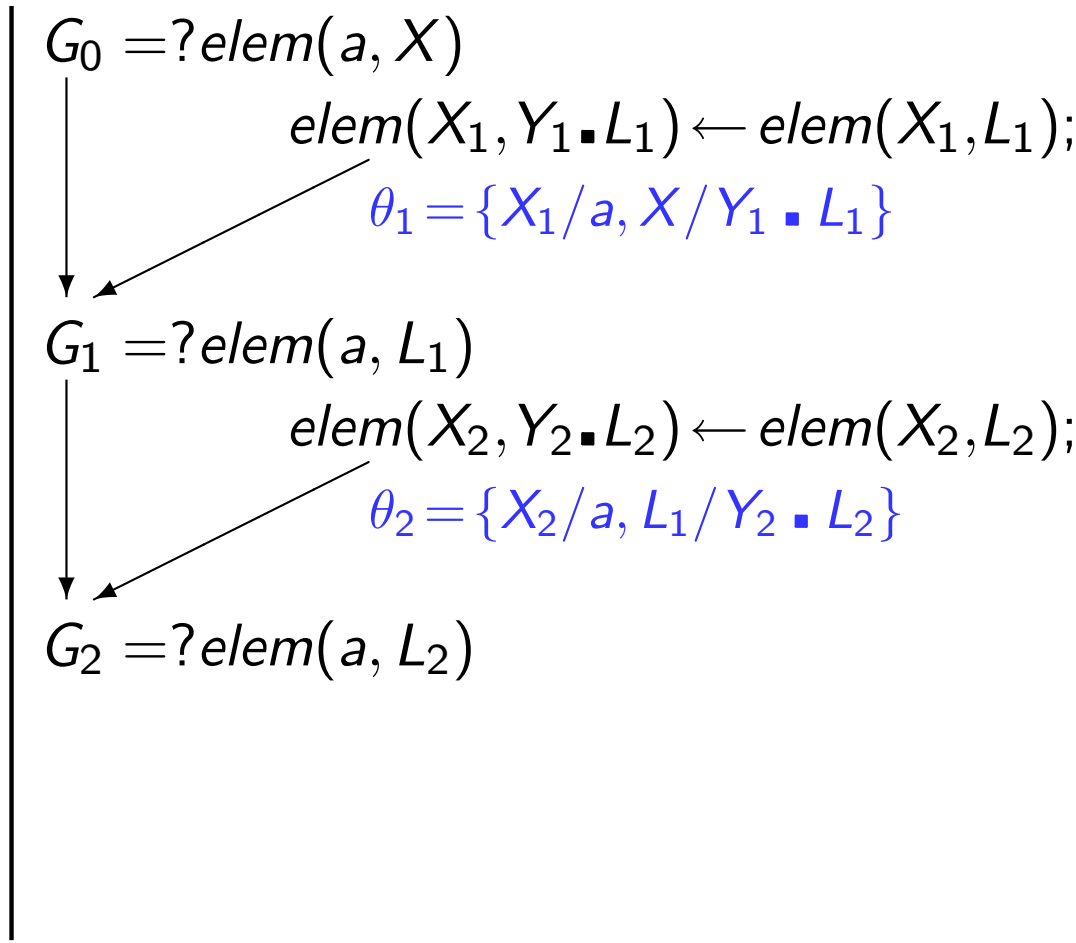
Логическая программа $\mathcal{P}$:

$elem(X, X \cdot L)$;

$elem(X, Y \cdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$



SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

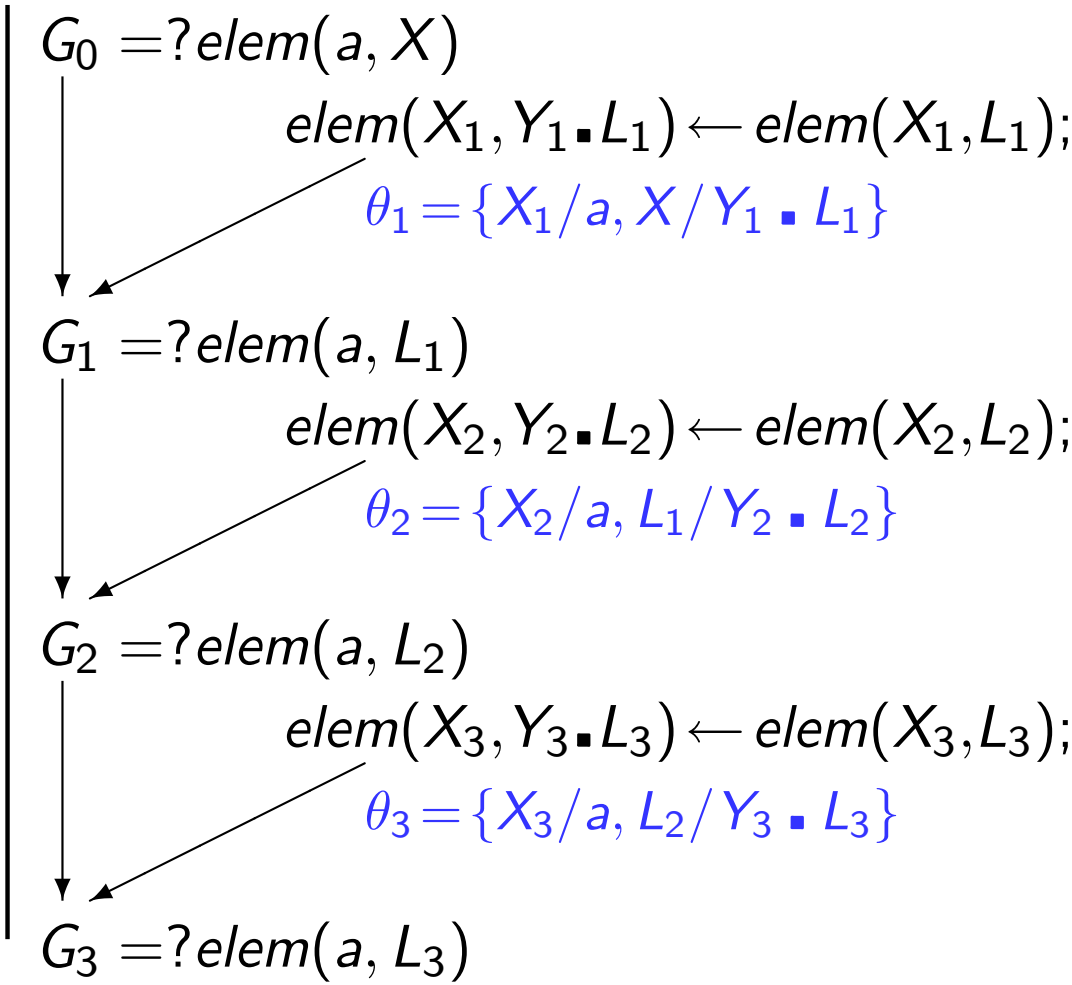
Логическая программа $\mathcal{P}$:

$elem(X, X \centerdot L)$;

$elem(X, Y \centerdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$



SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Пример 6.

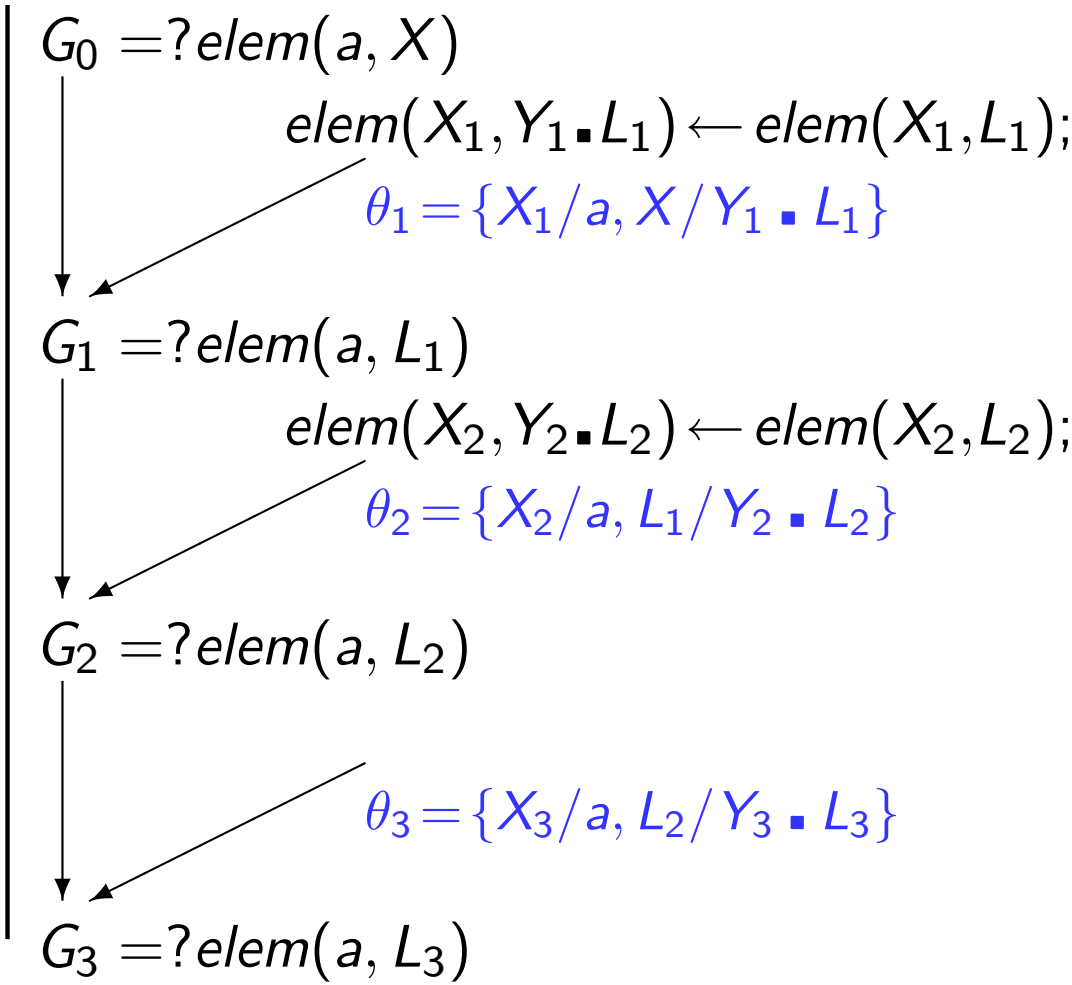
Логическая программа $\mathcal{P}$:

$elem(X, X \cdot L)$;

$elem(X, Y \cdot L) \leftarrow elem(X, L)$;

Запрос G_0:

? $elem(a, X)$



и. т. д. до ∞

Бесконечное вычисление.

SLD-РЕЗОЛЮТИВНЫЕ ВЫЧИСЛЕНИЯ

Теперь у нас есть два типа ответов на запросы к логическим программам:

- **правильные ответы**, которые логически следуют из программы;
- **вычисленные ответы**, которые конструируются по ходу SLD-резолютивных вычислений.

Правильные ответы — это то, что мы хотим получить, обращаясь с вопросами к программе.

Вычисленные ответы — это то, что нам в действительности выдает компьютер (интерпретатор программы).

Какова связь между правильными и вычисленными ответами?

КОНЕЦ ЛЕКЦИИ 12.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 13.

Корректность операционной семантики. Полнота операционной семантики логических программ.

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

У нас есть два типа ответов на запросы к логическим программам:

- **правильные ответы**, которые логически следуют из программы;
- **вычисленные ответы**, которые конструируются по ходу SLD-резолютивных вычислений.

Правильные ответы — это то, что мы хотим получить, обращаясь с вопросами к программе.

Вычисленные ответы — это то, что нам в действительности выдает компьютер (интерпретатор программы).

Какова связь между правильными и вычисленными ответами?

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Определение правильного ответа

Пусть $\mathcal{P}$ — логическая программа, G — запрос к $\mathcal{P}$ с множеством целевых переменных $Y_1, \ldots, Y_k$.

Тогда всякая подстановка $\theta = \{Y_1/t_1, \ldots, Y_k/t_k\}$ называется ответом на запрос G к программе $\mathcal{P}$.

Ответ $\theta = \{Y_1/t_1, \ldots, Y_k/t_k\}$ называется правильным ответом на запрос G к программе $\mathcal{P}$, если

$$\mathcal{P} \models \forall Z_1 \ldots \forall Z_N G\theta, \qquad \text{где } \{Z_1, \ldots, Z_N\} = \bigcup_{i=1}^{k} Var_{t_i}.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Определение вычисленного ответа

Пусть

- $G_0 = ?\ C_1, C_2, \ldots, C_m$ — целевое утверждение с целевыми переменными $Y_1, Y_2, \ldots, Y_k$,
- $\mathcal{P} = \{D_1, D_2, \ldots, D_N\}$ — хорновская логическая программа,
- $comp = (D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \ldots, (D_{j_n}, \theta_n, \square)$ — успешное SLD-резолютивное вычисление, порожденное запросом G к программе $\mathcal{P}$.

Тогда подстановка $\theta = (\theta_1\theta_2\ldots\theta_n)|_{Y_1, Y_2, \ldots, Y_k}$,

представляющая собой композицию всех вычисленных унификаторов $\theta_1,\ \theta_2, \ldots, \theta_n$, ограниченную целевыми переменными $Y_1, Y_2, \ldots, Y_k$,

называется вычисленным ответом на запрос G_0 к программе $\mathcal{P}$.

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Теорема (корректности операционной семантики)

Пусть

- $G_0 = ?\ C_1, C_2, \ldots, C_m$ — целевое утверждение,
- $\mathcal{P} = \{D_1, D_2, \ldots, D_N\}$ — хорновская логическая программа,
- θ — вычисленный ответ на запрос G_0 к программе $\mathcal{P}$.

Тогда θ — правильный ответ на запрос G_0 к программе $\mathcal{P}$.

Доказательство.

Рассмотрим успешное вычисление, порожденное запросом G_0 к логической программе $\mathcal{P}$:

$$comp\ =\ (D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \ldots, (D_{j_n}, \theta_n, \square)$$

Покажем индукцией по длине вычисления n, что
$\theta = (\theta_1\theta_2 \ldots \theta_n)|_{Y_1, Y_2, \ldots, Y_k}$ — это правильный ответ.

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Базис индукции ($n = 1$). Тогда

$$G_0 =? \; C_1 \qquad D_{j_1} = A_0; \qquad \theta_1 \in HOУ(C_1, A_0).$$

Значит, $C_1\theta_1 = A_0\theta_1$. Поскольку $D_{j_1} \in \mathcal{P}$, мы приходим к следующему выводу:

$$\mathcal{P} \models \forall \overline{X} A_0, \qquad \text{почему?}$$

и, следовательно,

$$\mathcal{P} \models \forall \overline{Z} A_0\theta_1, \qquad \text{почему?}$$

и, следовательно,

$$\mathcal{P} \models \forall \overline{Z} C_1\theta_1, \qquad \text{почему?}$$

и, следовательно, $\theta = \theta_1|_{Y_1, Y_2, \ldots, Y_k}$ — это правильный ответ.

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Индуктивный переход ($n-1 \rightarrow n$). Пусть

$$\begin{aligned} G_0 &= ?\ C_1, C_2, \ldots, \mathbf{C_i}, \ldots, C_m \\ D_{j_1} &= \mathbf{A_0} \leftarrow \mathbf{A_1}, \ldots, \mathbf{A_r}; \\ \theta_1 &\in HOY(\mathbf{C_i}, \mathbf{A_0}), \\ G_1 &= ?\ (C_1, C_2, \ldots, \mathbf{A_1}, \ldots, \mathbf{A_r}, \ldots, C_m)\theta_1. \end{aligned}$$

Тогда

$$comp' = (D_{j_2}, \theta_2, G_2), \ldots, (D_{j_n}, \theta_n, \Box)$$

— это успешное вычисление длины $n-1$, порожденное запросом G_1. Значит, по предположению индукции, $\theta' = \theta_2 \ldots \theta_n|_{Var_{G_1}}$ — правильный ответ на запрос G_1.

Значит,

$$\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Из $\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'$

следует

$$\mathcal{P} \models \forall \overline{Z}(A_1 \& \ldots \& A_r)\theta_1\theta'.$$

Поскольку $D_{j_1} \in \mathcal{P}$, верно

$$\mathcal{P} \models \forall \overline{X}(A_1 \& \ldots \& A_r \rightarrow A_0).$$

Значит, верно также

Таким образом,

$$\mathcal{P} \models \forall \overline{Z} A_0\theta_1\theta'.$$

И, наконец, вспомнив, что $A_0\theta_1 = C_i\theta_1$ (почему?), заключаем

$$\mathcal{P} \models \forall \overline{Z} C_i\theta_1\theta'.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Из $\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'$

следует

$$\mathcal{P} \models \forall \overline{Z}(A_1 \& \ldots \& A_r)\theta_1\theta'.$$

Поскольку $D_{j_1} \in \mathcal{P}$, верно

$$\mathcal{P} \models \forall \overline{X}(A_1 \& \ldots \& A_r \rightarrow A_0).$$

Значит, верно также

Таким образом,

$$\mathcal{P} \models \forall \overline{Z} A_0\theta_1\theta'.$$

И, наконец, вспомнив, что $A_0\theta_1 = C_i\theta_1$ (почему?), заключаем

$$\mathcal{P} \models \forall \overline{Z} C_i\theta_1\theta'.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Из $\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'$

следует

$$\mathcal{P} \models \forall \overline{Z}(A_1 \& \ldots \& A_r)\theta_1\theta'.$$

Поскольку $D_{j_1} \in \mathcal{P}$, верно

$$\mathcal{P} \models \forall \overline{X}(A_1 \& \ldots \& A_r \to A_0).$$

Значит, верно также

$$\mathcal{P} \models \forall \overline{Z}\Big((A_1 \& \ldots \& A_r)\theta_1\theta' \to A_0\theta_1\theta'\Big).$$

Таким образом,

$$\mathcal{P} \models \forall \overline{Z} A_0\theta_1\theta'.$$

И, наконец, вспомнив, что $A_0\theta_1 = C_i\theta_1$ (почему?), заключаем

$$\mathcal{P} \models \forall \overline{Z} C_i\theta_1\theta'.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Из $\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'$

следует

$$\mathcal{P} \models \forall \overline{Z}(A_1 \& \ldots \& A_r)\theta_1\theta'.$$

Поскольку $D_{j_1} \in \mathcal{P}$, верно

$$\mathcal{P} \models \forall \overline{X}(A_1 \& \ldots \& A_r \to A_0).$$

Значит, верно также

$$\mathcal{P} \models \forall \overline{Z}\Big((A_1 \& \ldots \& A_r)\theta_1\theta' \to A_0\theta_1\theta'\Big).$$

Таким образом,

$$\mathcal{P} \models \forall \overline{Z} A_0\theta_1\theta'.$$

И, наконец, вспомнив, что $A_0\theta_1 = C_i\theta_1$ (почему?), заключаем

$$\mathcal{P} \models \forall \overline{Z} C_i\theta_1\theta'.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Из $\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'$

следует

$$\mathcal{P} \models \forall \overline{Z}(A_1 \& \ldots \& A_r)\theta_1\theta'.$$

Поскольку $D_{j_1} \in \mathcal{P}$, верно

$$\mathcal{P} \models \forall \overline{X}(A_1 \& \ldots \& A_r \to A_0).$$

Значит, верно также

$$\mathcal{P} \models \forall \overline{Z}\Big((A_1 \& \ldots \& A_r)\theta_1\theta' \to A_0\theta_1\theta'\Big).$$

Таким образом,

$$\mathcal{P} \models \forall \overline{Z} A_0\theta_1\theta'.$$

И, наконец, вспомнив, что $A_0\theta_1 = C_i\theta_1$ (почему?), заключаем

$$\mathcal{P} \models \forall \overline{Z} C_i\theta_1\theta'.$$

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство.

Итак, из $\mathcal{P} \models \forall \overline{Z}(C_1 \& C_2 \& \ldots \& A_1 \& \ldots \& A_r \& \ldots \& C_m)\theta_1\theta'$

следует

$$\mathcal{P} \models \forall \overline{Z}(C_1 \& \ldots \& C_{i-1} \& C_{i+1} \& \ldots C_m)\theta_1\theta'.$$

Мы показали, что

$$\mathcal{P} \models \forall \overline{Z}\, C_i\theta_1\theta'.$$

Значит,

$$\mathcal{P} \models \forall \overline{Z}(\underbrace{C_1 \& \ldots \& C_{i-1} \& C_i \& C_{i+1} \& \ldots C_m}_{G_0})\underbrace{\theta_1\theta'}_{\theta}.$$

Значит, $\theta_1\theta'|_{Var_{G_0}} = (\theta_1\theta_2\ldots\theta_n)|_{Y_1, Y_2, \ldots, Y_k} = \theta$ — правильный ответ на запрос G_0. □

КОРРЕКТНОСТЬ ОПЕРАЦИОННОЙ СЕМАНТИКИ

Итак, всякий вычисленный ответ — правильный.

А можно ли вычислить любой правильный ответ?

Полна ли наша операционная семантика?

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ ЛОГИЧЕСКИХ ПРОГРАММ

Проблема полноты

Пусть θ — правильный ответ на запрос G_0 к хорновской логической программе $\mathcal{P}$, т. е. $\mathcal{P} \models \forall \overline{Z}\ G_0\theta$.

Существует ли такое успешное SLD-резолютивное вычисление, порожденное запросом G_0 к программе $\mathcal{P}$

$$(D_{j_1}, \eta_1, G_1), (D_{j_2}, \eta_2, G_2), \ldots, (D_{j_n}, \eta_n, \square),$$

которое вычисляет ответ θ, т. е. $\theta = (\eta_1\eta_2 \ldots \eta_n)|_{Var_{G_0}}$?

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ ЛОГИЧЕСКИХ ПРОГРАММ

Теорема полноты.

Пусть

$\mathcal{P} = \{D_1, D_2, \ldots, D_N\}$ — хорновская логическая программа,

$G_0 = ?C_1, C_2, \ldots C_m$ — запрос с множеством целевых переменных $Y_1, Y_2, \ldots, Y_k$,

θ — правильный ответ на запрос G_0 к хорновской логической программе $\mathcal{P}$.

Тогда существует такой вычисленный ответ η на запрос G_0 к программе $\mathcal{P}$, что

$$\theta = \eta\rho$$

для некоторой подстановки ρ.

Теорема полноты гласит, что каждый правильный ответ — это пример (частный случай) некоторого вычисленного ответа .

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ ЛОГИЧЕСКИХ ПРОГРАММ

Доказательство.

Пусть $\theta = \{Y_1/t_1, \ldots, Y_k/t_k\}$ и пусть $\bigcup_{i=1}^{k} Var_{t_i} = \{Z_1, \ldots, Z_r\}$.

Выберем некоторое множество новых «свежих» констант $\{c_1, \ldots, c_k\}$, **не содержащихся** ни в программе $\mathcal{P}$, ни в запросе G, ни в термах подстановки θ, и рассмотрим подстановку $\lambda = \{Z_1/c_1, Z_2/c_2, \ldots, Z_r/c_r\}$.

Поскольку θ — правильный ответ, верно $\mathcal{P} \models \forall Z_1 \ldots \forall Z_k G_0\theta$. Запрос $G_0' = G_0\theta\lambda$ — основной пример запроса G_0, и поэтому $\mathcal{P} \models G_0\theta\lambda$.

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ ЛОГИЧЕСКИХ ПРОГРАММ

Доказательство.

Общий замысел доказательства

1. Вначале покажем, что запрос $G_0' = G_0\theta\lambda$, обращенный к множеству $[\mathcal{P}]$ основных примеров программных утверждений программы $\mathcal{P}$, имеет успешное SLD-резолютивное вычисление (лемма об основных вычислениях).

2. Затем покажем, что исходный запрос G_0, обращенный к самой программе $\mathcal{P}$, имеет успешное SLD-резолютивное вычисление с таким вычисленным ответом η, для которого верно равенство $\theta\lambda = \eta\rho'$ для некоторой подстановки ρ' (лемма о подъеме для хорновских дизъюнктов).

3. Затем покажем, что отсюда следует равенство $\theta = \eta\rho$ для некоторой подстановки ρ.

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Лемма об основных вычислениях.

Пусть

$\mathcal{P} = \{D_1, D_2, \ldots, D_N\}$ — хорновская логическая программа,

$G'_0 = ?C'_1, C'_2, \ldots C'_n$ — **основной** запрос,

и при этом верно $\mathcal{P} \models G'_0$.

Тогда существует успешное SLD-резолютивное вычисление запроса G'_0, обращенного к множеству $[\mathcal{P}]$ основных примеров программных утверждений программы $\mathcal{P}$.

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

$\{D_1, D_2, \ldots, D_N\} \models C'_1 \& C'_2 \& \ldots \& C'_n$

$\Longleftrightarrow$

множество дизъюнктов $\{D_1, D_2, \ldots, D_N, \neg C'_1 \vee \neg C_2 \vee \ldots \vee \neg C'_n\}$ противоречиво.

$\Longleftrightarrow$ (по теореме Эрбрана)

существует такое конечное множество основных примеров программных утверждений $\{D'_1, D'_2, \ldots, D'_M\} \subseteq [\mathcal{P}]$, что множество дизъюнктов $\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}$ противоречиво.

$\Longleftrightarrow$ (по теореме полноты метода резолюций)

из системы дизъюнктов $\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \ldots \vee \neg C'_n\}$ резолютивно выводим пустой дизъюнкт $\square$.

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Да, пустой дизъюнкт $\square$ можно резолютивно вывести из системы

$$\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}.$$

Но это не обязательно SLD-резолютивный вывод. Рассмотрим его более подробно.

смешанные дизъюнкты

$$D'_1 = A_{01} \vee \neg A_{11} \vee \cdots \vee \neg A_{k1}$$

$$D'_2 = A_{02} \vee \neg A_{12} \vee \cdots \vee \neg A_{r2}$$

$$\cdots$$

$$D'_M = A_{0M} \vee \neg A_{1M} \vee \cdots \vee \neg A_{\ell M}$$

негативные дизъюнкты

$$G'_0 = \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n$$

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Да, пустой дизъюнкт $\square$ можно резолютивно вывести из системы

$$\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}.$$

Но это не обязательно SLD-резолютивный вывод. Рассмотрим его более подробно.

смешанные дизъюнкты

$$D'_1 = A_{01} \vee \neg A_{11} \vee \cdots \vee \neg A_{k1}$$

$$D'_2 = A_{02} \vee \neg A_{12} \vee \cdots \vee \neg A_{r2}$$

$$\cdots$$

$$D'_M = A_{0M} \vee \neg A_{1M} \vee \cdots \vee \neg A_{\ell M}$$

негативные дизъюнкты

$$G'_0 = \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n$$

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Да, пустой дизъюнкт $\square$ можно резолютивно вывести из системы

$$\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}.$$

Но это не обязательно SLD-резолютивный вывод. Рассмотрим его более подробно.

смешанные дизъюнкты

$$D'_1 = A_{01} \vee \neg A_{11} \vee \cdots \vee \neg A_{k1}$$

$$D'_2 = A_{02} \vee \neg A_{12} \vee \cdots \vee \neg A_{r2}$$

$$\cdots$$

$$D'_M = A_{0M} \vee \neg A_{1M} \vee \cdots \vee \neg A_{\ell M}$$

негативные дизъюнкты

$$G'_0 = \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n$$

возможны резольвенты двух типов

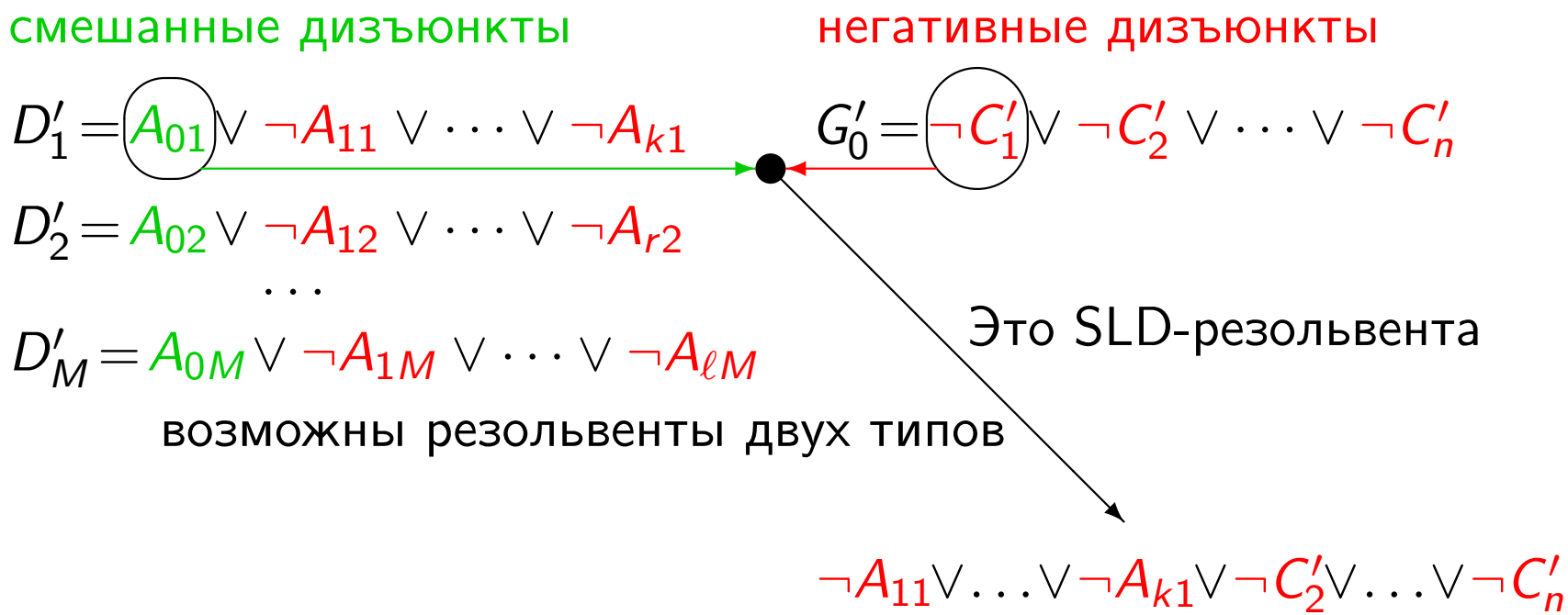
ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Да, пустой дизъюнкт $\square$ можно резолютивно вывести из системы

$$\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}.$$

Но это не обязательно SLD-резолютивный вывод. Рассмотрим его более подробно.



ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Да, пустой дизъюнкт $\square$ можно резолютивно вывести из системы

$$\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}.$$

Но это не обязательно SLD-резолютивный вывод. Рассмотрим его более подробно.

смешанные дизъюнкты

негативные дизъюнкты

$D'_1 = A_{01} \vee \neg A_{11} \vee \cdots \vee \neg A_{k1}$

$G'_0 = \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n$

$D'_2 = A_{02} \vee \neg A_{12} \vee \cdots \vee \neg A_{r2}$

$\ldots$

$D'_M = A_{0M} \vee \neg A_{1M} \vee \cdots \vee \neg A_{\ell M}$

возможны резольвенты двух типов

А это не SLD-резольвента

$\neg A_{11} \vee \ldots \vee \neg A_{k1} \vee \neg C'_2 \vee \ldots \vee \neg C'_n$

$A_{02} \vee \neg A_{22} \vee \ldots \vee \neg A_{r2} \vee \neg A_{11} \vee \ldots \vee \neg A_{k1}$

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Да, пустой дизъюнкт $\square$ можно резолютивно вывести из системы

$$\{D'_1, D'_2, \ldots, D'_M, \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n\}.$$

Но это не обязательно SLD-резолютивный вывод. Рассмотрим его более подробно.

смешанные дизъюнкты

$$D'_1 = A_{01} \vee \neg A_{11} \vee \cdots \vee \neg A_{k1}$$

$$D'_2 = A_{02} \vee \neg A_{12} \vee \cdots \vee \neg A_{r2}$$

$$\cdots$$

$$D'_M = A_{0M} \vee \neg A_{1M} \vee \cdots \vee \neg A_{\ell M}$$

негативные дизъюнкты

$$G'_0 = \neg C'_1 \vee \neg C'_2 \vee \cdots \vee \neg C'_n$$

возможны резольвенты двух типов

$$\neg A_{11} \vee \ldots \vee \neg A_{k1} \vee \neg C'_2 \vee \ldots \vee \neg C'_n$$

$$A_{02} \vee \neg A_{22} \vee \ldots \vee \neg A_{r2} \vee \neg A_{11} \vee \ldots \vee \neg A_{k1}$$

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Покажем, что этот вывод можно перестроить так, чтобы в нем остались только SLD-резольвенты. В этом выводе обязательно есть хотя бы одна SLD-резольвента (почему?),

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

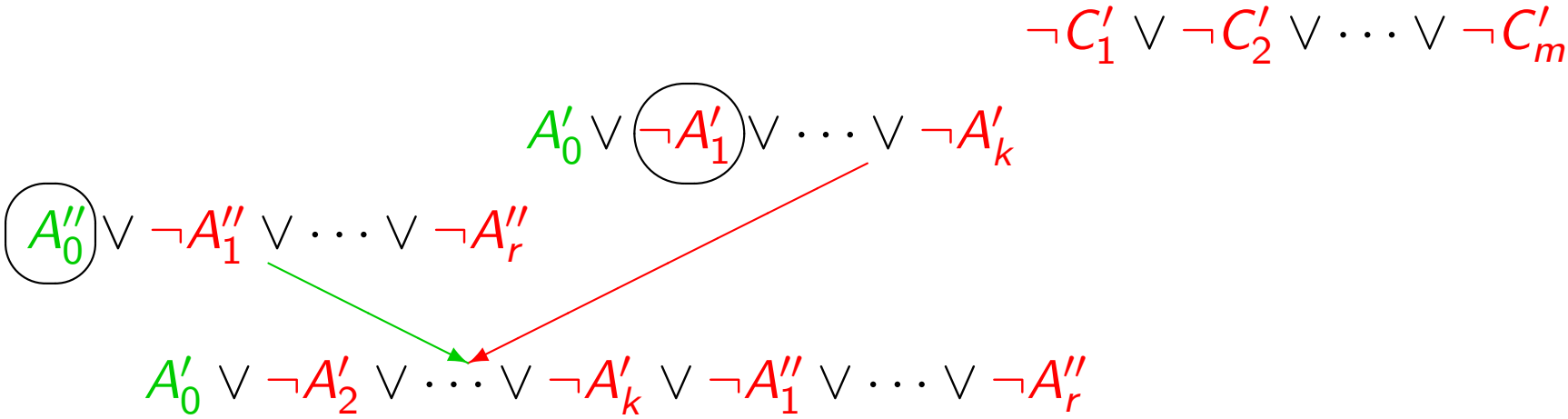
Доказательство леммы о вычислениях.

Покажем, что этот вывод можно перестроить так, чтобы в нем остались только SLD-резольвенты. В этом выводе обязательно есть хотя бы одна SLD-резольвента (почему?), потому что пустой дизъюнкт — это всегда SLD-резольвента (почему?).

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Покажем, что этот вывод можно перестроить так, чтобы в нем остались только SLD-резольвенты. В этом выводе обязательно есть хотя бы одна SLD-резольвента (почему?), потому что пустой дизъюнкт — это всегда SLD-резольвента (почему?). Тогда будем поступать так:

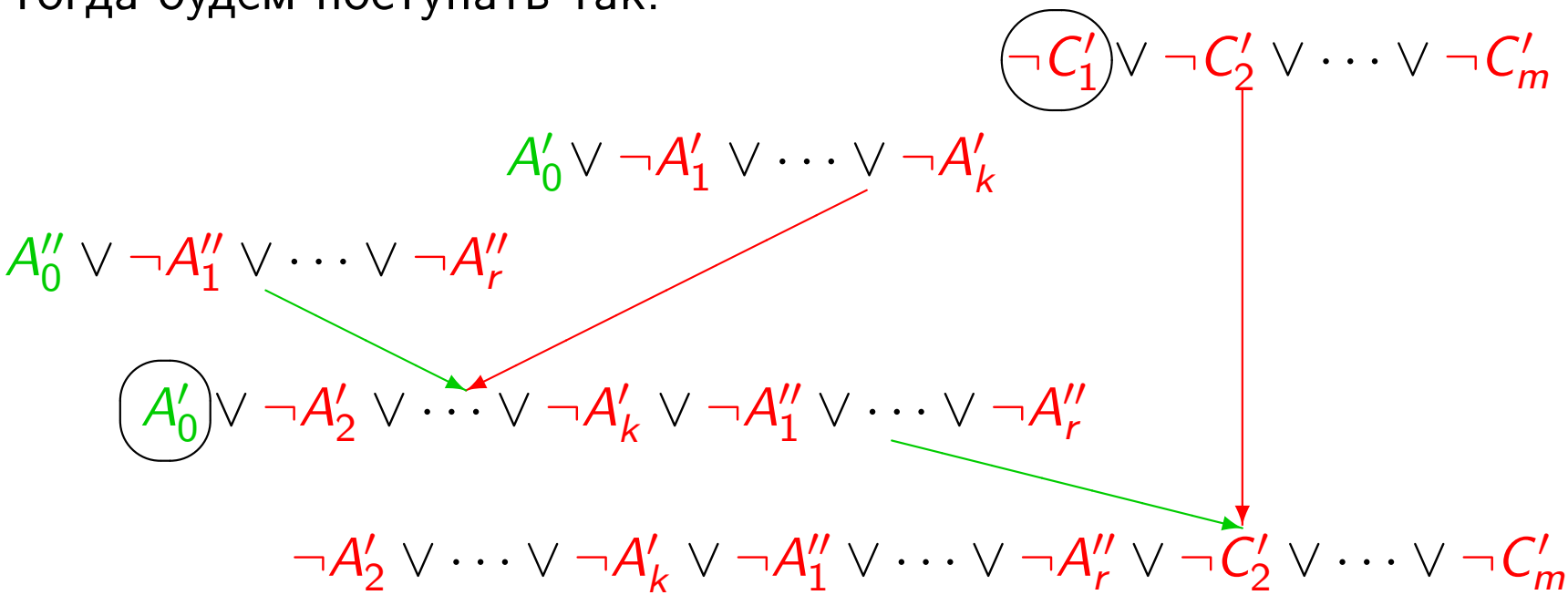


Если правило резолюции вначале применяется к двум программным утверждениям,

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Покажем, что этот вывод можно перестроить так, чтобы в нем остались только SLD-резольвенты. В этом выводе обязательно есть хотя бы одна SLD-резольвента (почему?), потому что пустой дизъюнкт — это всегда SLD-резольвента (почему?). Тогда будем поступать так:

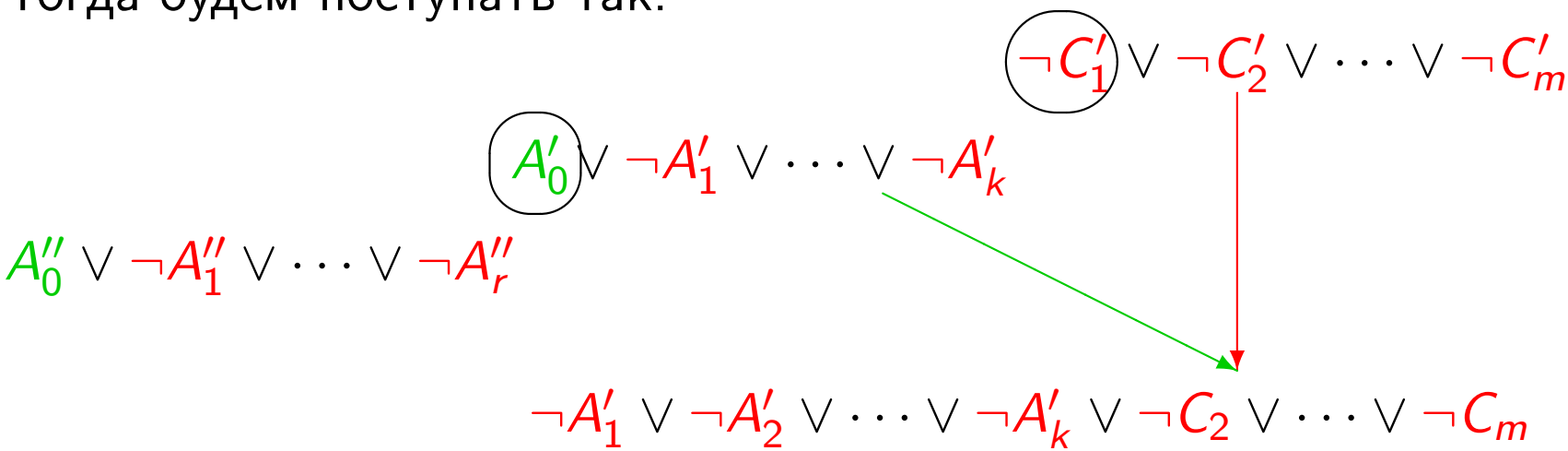


а затем применяется к полученной резольвенте и запросу,

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Покажем, что этот вывод можно перестроить так, чтобы в нем остались только SLD-резольвенты. В этом выводе обязательно есть хотя бы одна SLD-резольвента (почему?), потому что пустой дизъюнкт — это всегда SLD-резольвента (почему?). Тогда будем поступать так:

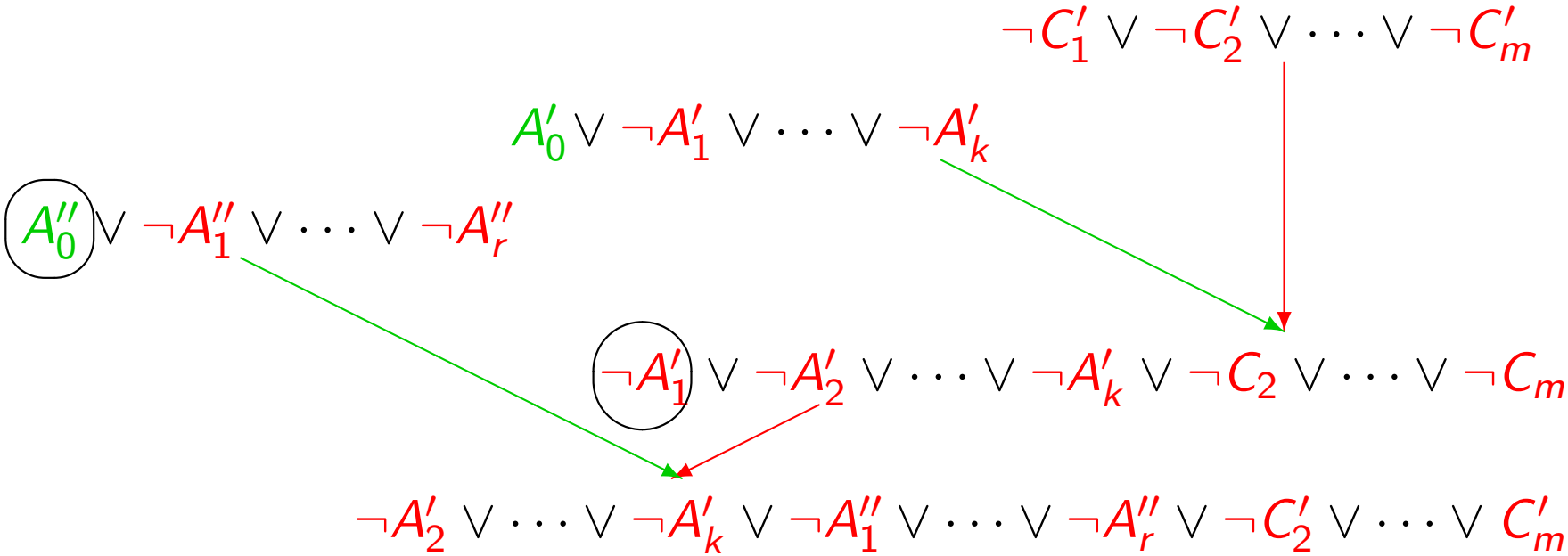


то изменим порядок применения правил

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство леммы о вычислениях.

Покажем, что этот вывод можно перестроить так, чтобы в нем остались только SLD-резольвенты. В этом выводе обязательно есть хотя бы одна SLD-резольвента (почему?), потому что пустой дизъюнкт — это всегда SLD-резольвента (почему?). Тогда будем поступать так:



и получим тот же самый результат,

но уже только при помощи SLD-резолюции.

ЛЕММА ОБ ОСНОВНЫХ ВЫЧИСЛЕНИЯХ

Доказательство.

Будем применять этот прием, до тех пор пока в выводе не останутся только правила SLD-резолюции. Таким образом, в итоге получим вывод пустого дизъюнкта $\square$ из множества дизъюнктов

$$\{D_1, D_2, \ldots, D_N, \neg C_1' \vee \neg C_2 \vee \cdots \vee \neg C_n'\}$$

только при помощи правила SLD-резолюции.

Это и есть успешное SLD-резолютивное вычисление основного запроса $G_0' = ?C_1', C_2', \ldots C_n'$, обращенного к множеству основных примеров программных утверждений $[\mathcal{P}]$.

Что и требовалось доказать. $\square$

ЛЕММА О ПОДЪЕМЕ

Лемма о подъеме (для логических программ)

Пусть $G_0' = G_0\theta'$ — основной пример запроса G_0 с множеством целевых переменных $Y_1, \ldots, Y_m$, обращенный к хорновской логической программе $\mathcal{P}$.

Если запрос G_0', обращенный к множеству **основных примеров** программных утверждений $[\mathcal{P}]$, имеет успешное вычисление,

то исходный запрос G_0, обращенный к самой программе $\mathcal{P}$, также имеет успешное вычисление с ответом η, который удовлетворяет равенству

$$\theta' = \eta\rho'$$

для некоторой подстановки ρ'.

ЛЕММА О ПОДЪЕМЕ

Доказательство леммы о подъеме

Рассмотрим SLD-резолютивное вычисление запроса $G'_0 = G_0\theta'$ с использованием основных примеров программных утверждений из множества $[\mathcal{P}]$

$$(D'_1, \varepsilon, G'_1), (D'_2, \varepsilon, G'_2), \ldots, (D'_n, \varepsilon, \square)$$

и покажем, что существует SLD-резолютивное вычисление запроса G_0 с использованием программы $\mathcal{P}$

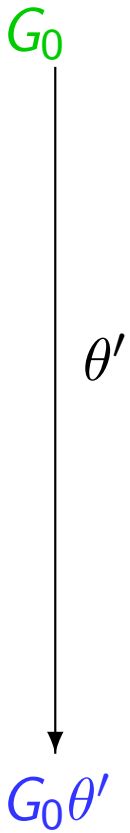
$$(D_1, \eta_1, G_1), (D_2, \eta_2, G_2), \ldots, (D_n, \eta_n, \square),$$

удовлетворяющее условиям леммы.

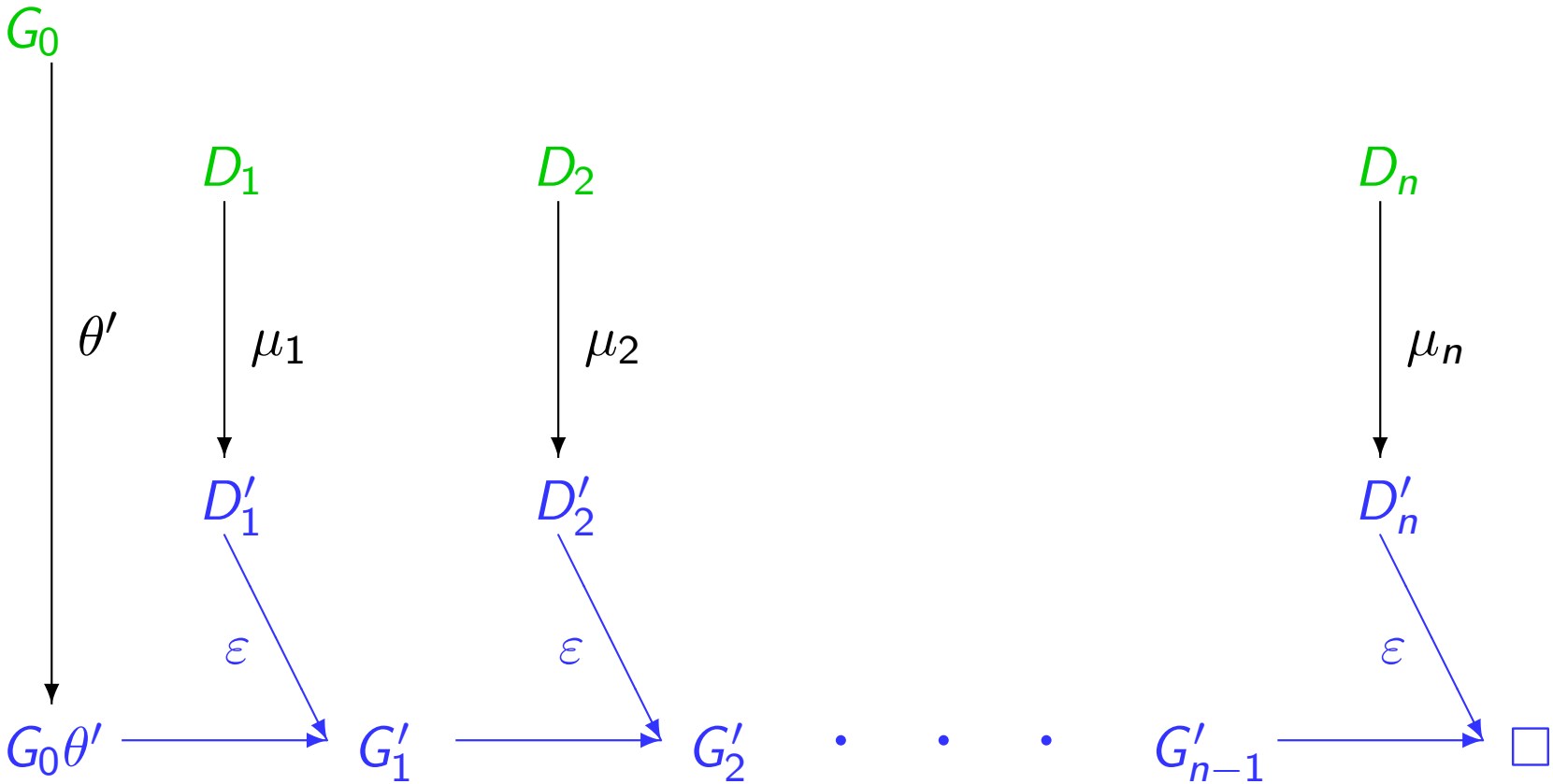
ЛЕММА О ПОДЪЕМЕ

G_0

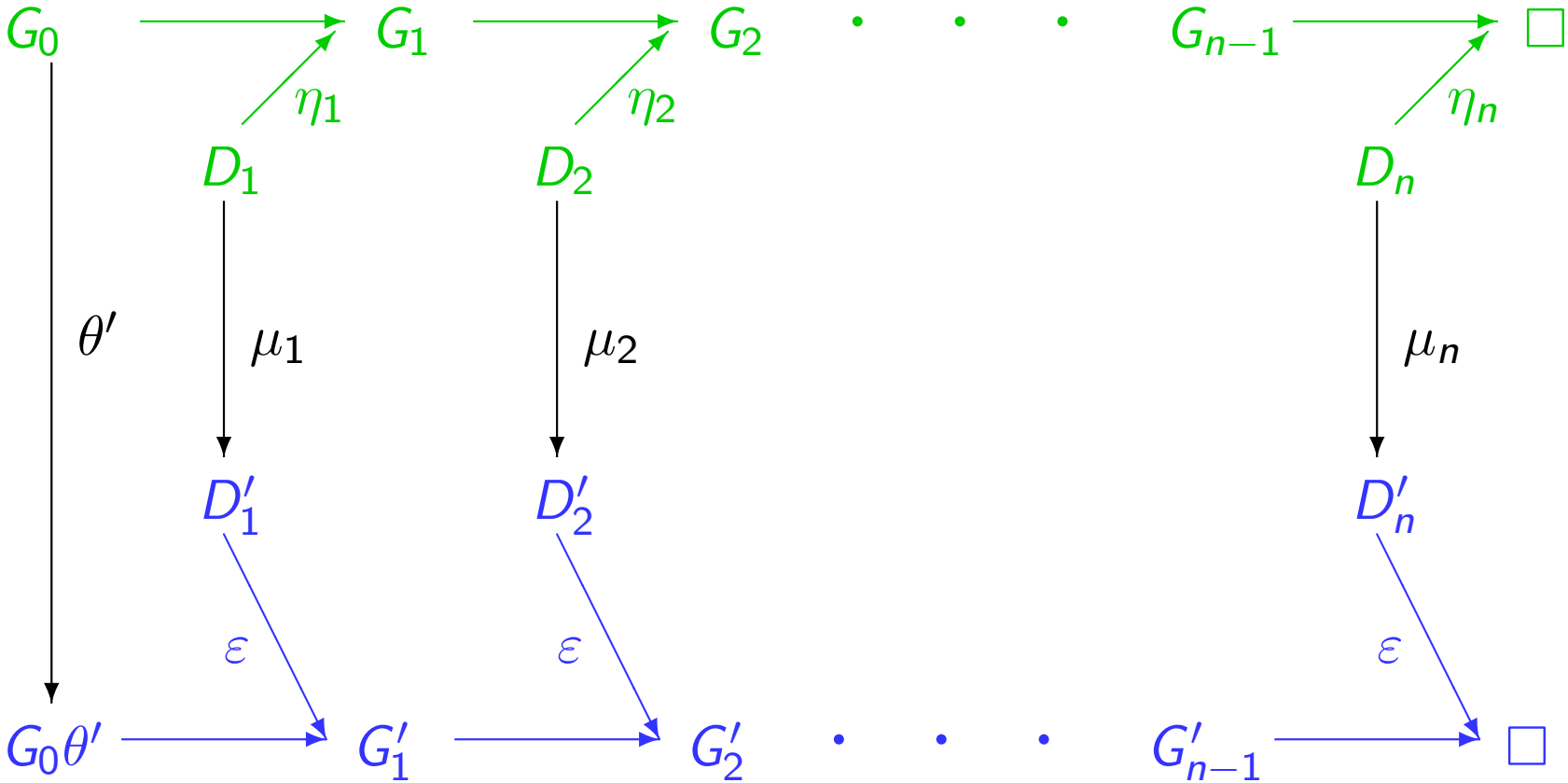
ЛЕММА О ПОДЪЕМЕ



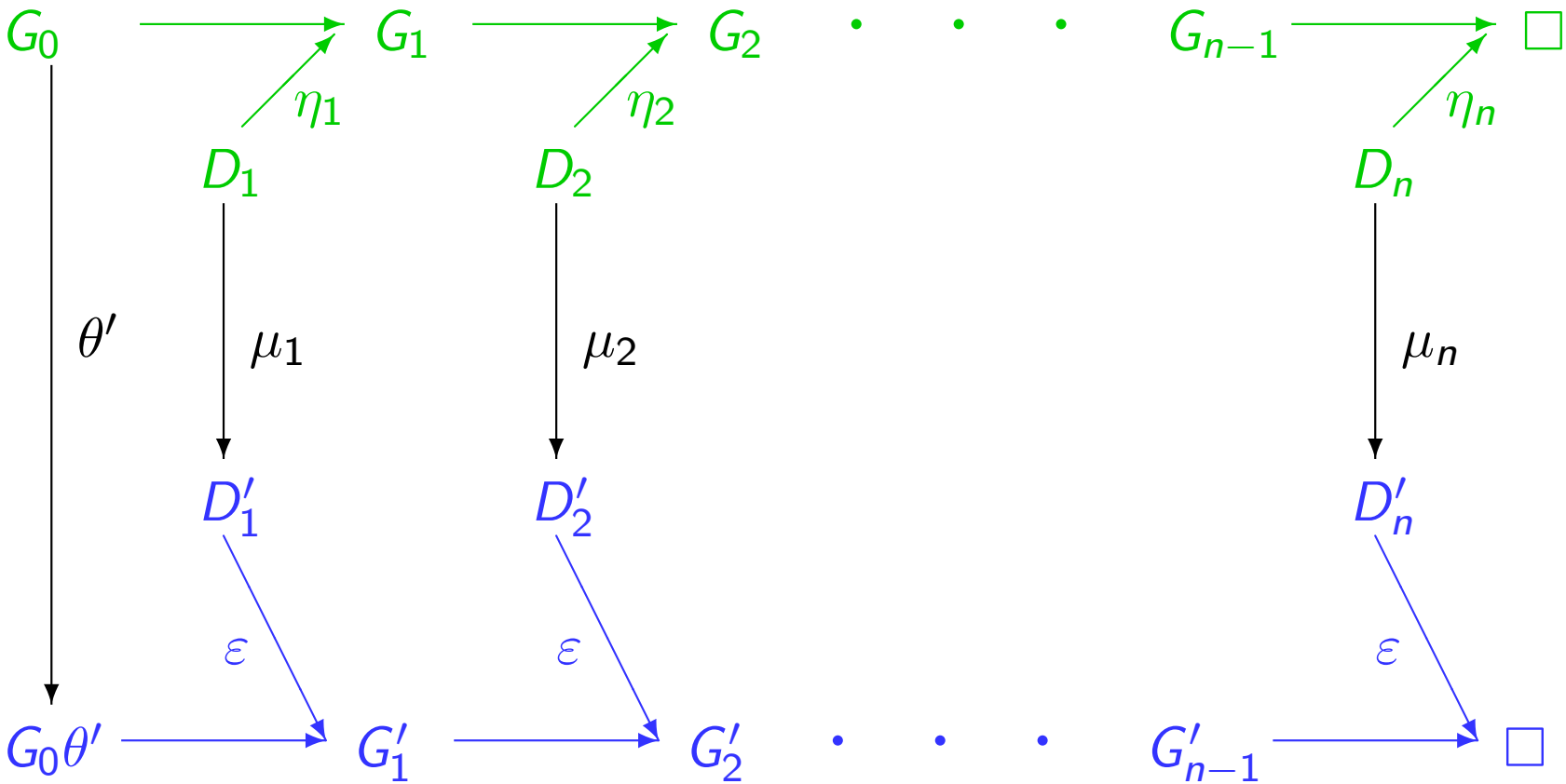
ЛЕММА О ПОДЪЕМЕ



ЛЕММА О ПОДЪЕМЕ



ЛЕММА О ПОДЪЕМЕ

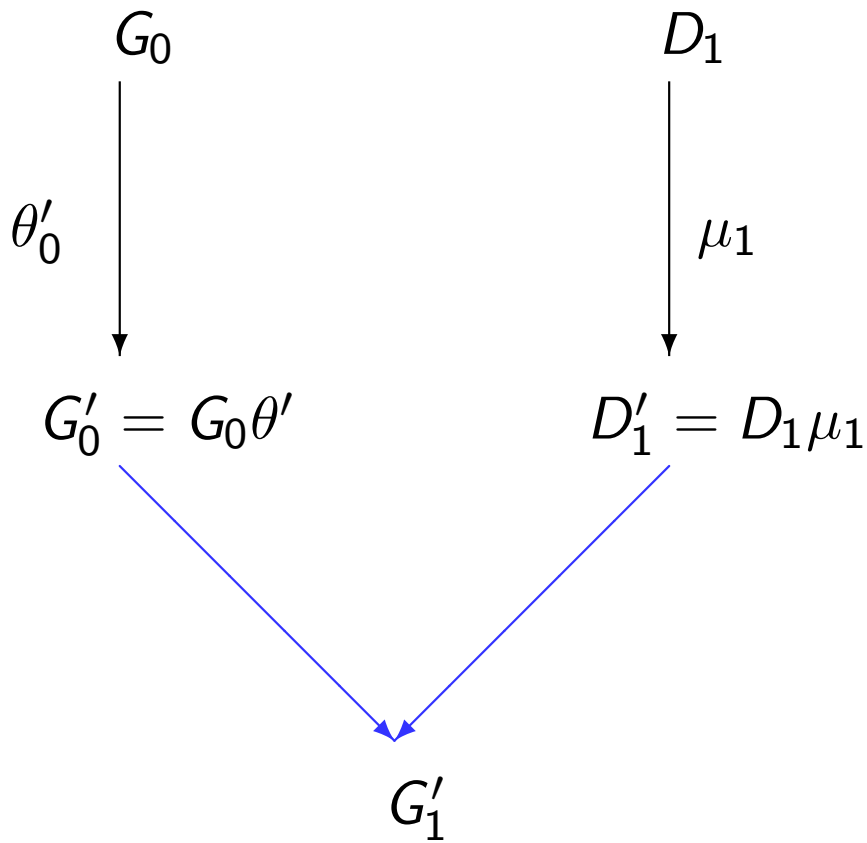


$$\theta' = (\eta_1\eta_2\ldots\eta_n)|_{Y_1,\ldots,Y_m}\rho'$$

ЛЕММА О ПОДЪЕМЕ

Доказательство леммы о подъеме

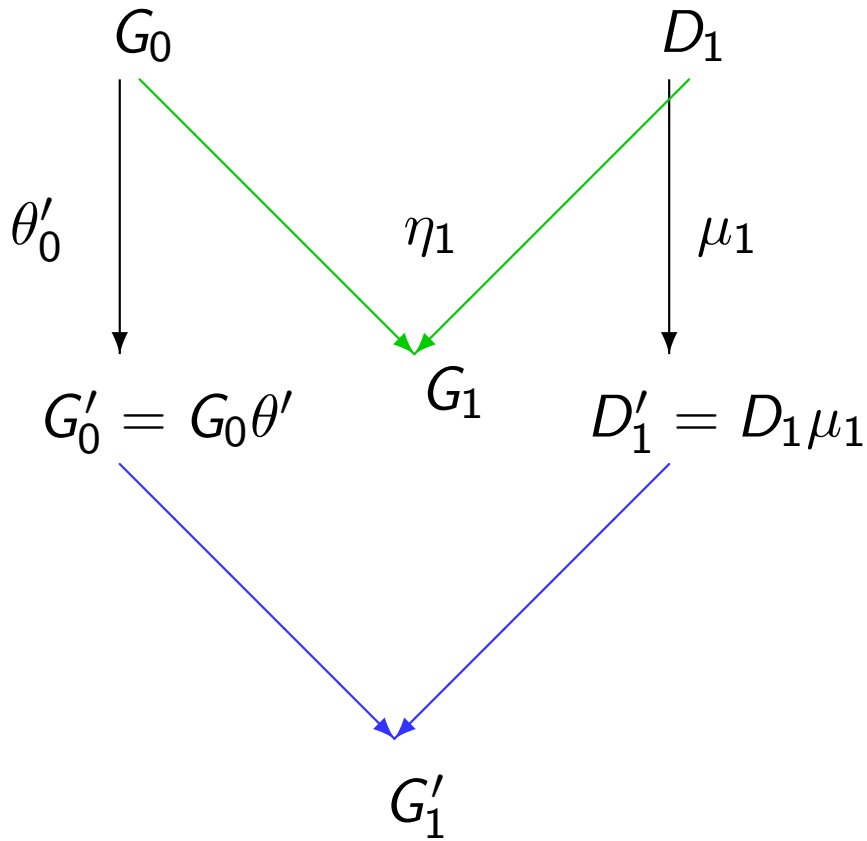
Воспользуемся леммой о подъеме для обычных дизъюнктов.



ЛЕММА О ПОДЪЕМЕ

Доказательство леммы о подъеме

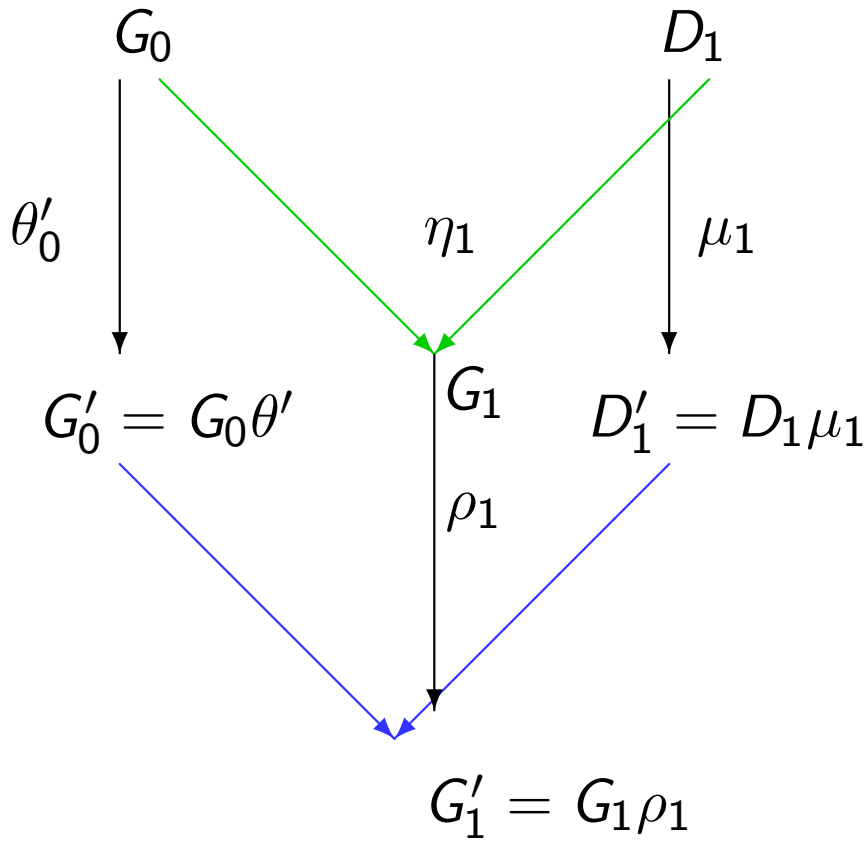
Воспользуемся леммой о подъеме для обычных дизъюнктов.



ЛЕММА О ПОДЪЕМЕ

Доказательство леммы о подъеме

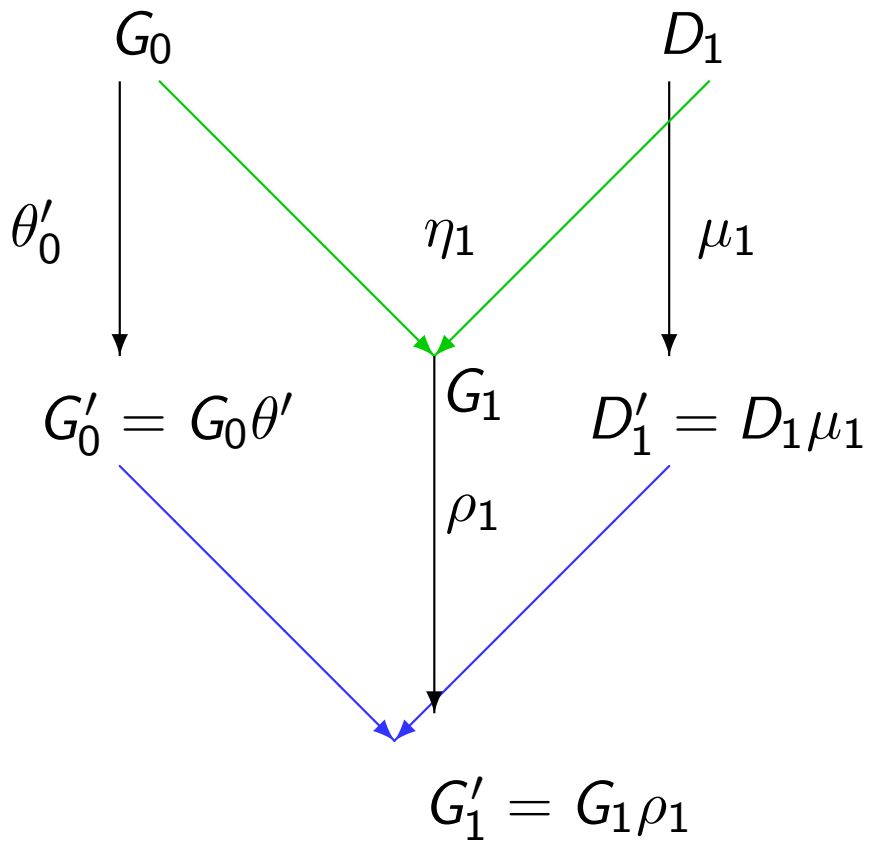
Воспользуемся леммой о подъеме для обычных дизъюнктов.



ЛЕММА О ПОДЪЕМЕ

Доказательство леммы о подъеме

Воспользуемся леммой о подъеме для обычных дизъюнктов.



И при этом верно равенство $\theta'_0 = (\eta_1\rho_1)|_{Y_1,\ldots,Y_m}$.

ЛЕММА О ПОДЪЕМЕ

Доказательство леммы о подъеме

Последовательно применяя этот прием на всех шагах вычисления запроса G'_0, получаем SLD-резолютивное вычисление запроса G_0:

$$(D_1, \eta_1, G_1), (D_2, \eta_1, G_2), \ldots, (D_n, \eta_n, \square)$$

для которого выполняется система равенств

$$\begin{aligned}
\theta' &= (\eta_1\rho_1)|_{Y_1,\ldots,Y_m} \\
\rho_1 &= (\eta_2\rho_2)|_{Var_{G_1}} \\
\rho_2 &= (\eta_3\rho_3)|_{Var_{G_2}} \\
&\ldots \\
\rho_n &= (\eta_n\rho_n)|_{Var_{G_n}}
\end{aligned}$$

Из этой системы следует равенство $\theta' = (\eta_1\eta_2\ldots\eta_n)|_{Y_1,\ldots,Y_m}\rho'$ для некоторой подстановки ρ'. $\square$

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство теоремы полноты (завершение).

Итак, у нас есть

- правильный ответ θ_0 на запрос G_0 к хорновской логической программе $\mathcal{P}$;
- подстановка $\lambda = \{Z_1/c_1, Z_2/c_2, \ldots, Z_r/c_r\}$ «свежих» констант на место всех переменных $Z_1, \ldots, Z_r$ из термов подстановки θ_0.
- основной пример $\theta'_0 = \theta_0\lambda$ правильного ответа θ.

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство теоремы полноты (завершение).

θ_0 — правильный ответ на запрос G_0 к хорновской логической программе $\mathcal{P}$;

$\Downarrow$

$\mathcal{P} \models G_0\theta_0\lambda$;

$\Downarrow$ (по лемме об основных вычислениях)

основной запрос $G_0\theta_0\lambda$ к множеству основных примеров программных утверждений $[\mathcal{P}]$ имеет успешное вычисление;

$\Downarrow$ (по лемме о подъеме для логических программ)

запрос G_0 к программе $\mathcal{P}$ имеет успешное вычисление с вычисленным ответом η, для которого верно равенство

$$\theta_0\lambda = \eta\rho'$$

для некоторой подстановки ρ'.

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Доказательство теоремы полноты (завершение).

А теперь заменим в левой и правой частях равенства

$$\theta_0\lambda = \eta\rho'$$

все константы $c_1, \ldots, c_r$ на символы переменных $Z_1, \ldots, Z_r$.

Поскольку константы $c_1, \ldots, c_r$ не входят в состав запроса G_0 и программы $\mathcal{P}$, эти константы не входят в состав термов вычисленного ответа η, и, значит, могут содержаться только в подстановке ρ'.

В левой части равенства подстановка $\lambda = \{Z_1/c_1, \ldots, Z_r/c_r\}$ превращается в пустую подстановку ε.

В правой части равенства подстановка ρ' превращается в некоторую новую подстановку ρ.
В итоге равенство $\theta_0\lambda = \eta\rho'$ превращается в равенство

$$\theta_0 = \eta\rho$$

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Поясняющий пример.

Рассмотрим запрос $?P(U, V)$ к логической программе

$$\begin{array}{lll} \mathcal{P}: & P(f(X), X) \leftarrow R(X); & (1) \\ & R(Y) \leftarrow; & (2) \\ & Q(c) \leftarrow; & (3) \end{array}$$

Легко видеть, что $\theta = \{U/f(c), V/c\}$ — это правильный ответ на запрос к программе.

Вместе с тем, единственный вычисленный ответ — это $\eta = \{U/f(Y), V/Y\}$.

Все дело в том, что θ — это частный случай η: $\theta = \eta\{Y/c\}$.

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Итак, любой правильный ответ на запрос к хорновской логической программе можно вычислить (возможно, с обобщением) по правилу SLD-резолюции, и любой вычисленный ответ будет правильным.

А как организовать вычисления логических программ, чтобы вычислить **ВСЕ** ответы?

КОНЕЦ ЛЕКЦИИ 13.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 14.

Правила выбора подцелей. Деревья вычислений логических программ. Стратегии вычисления логических программ.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

SLD-резолютивное вычисление запроса $?G$ к логической программе $\mathcal{P}$ определяется неоднозначно.
Рассмотрим запрос $?P(U,V), R(U)$ к логической программе

$$\begin{array}{lll} \mathcal{P}: & P(X,Y) \leftarrow R(X), Q(Y); & (1) \\ & P(X,X) \leftarrow Q(X); & (2) \\ & R(b) \leftarrow; & (3) \\ & Q(c) \leftarrow; & (4) \end{array}$$

Уже на первом шаге вычисления возникают вопросы выбора:

- Какую подцель выделить?

$$?P(U,V), R(U) \qquad\qquad ?P(U,V), R(U)$$

- Если выделена, например, первая подцель $?P(U,V), R(U)$, то какое программное утверждение выбрать?

$$\begin{array}{lll} P(X,Y) \leftarrow R(X), Q(Y); & (1) \\ P(X,X) \leftarrow Q(X); & (2) \end{array} \qquad \begin{array}{lll} P(X,Y) \leftarrow R(X), Q(Y); & (1) \\ P(X,X) \leftarrow Q(X); & (2) \end{array}$$

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Рассмотрим два вычисления запроса $?P(U, V), R(U)$

$?P(U, V), R(U)$

$P(X_1, Y_1) \leftarrow R(X_1), Q(Y_1)$

$\theta_1 = \{U/X_1, V/Y_1\}$

$?R(X_1), Q(Y_1), R(X_1)$

$R(b) \leftarrow$

$\theta_2 = \{X_1/b\}$

$?Q(Y_1), R(b)$

$Q(c) \leftarrow$

$\theta_3 = \{Y_1/c\}$

$?R(b)$

$R(b) \leftarrow$

$\theta_4 = \varepsilon$

$?\square$

$?P(U, V), R(U)$

$R(b) \leftarrow$

$\eta_1 = \{U/b\}$

$?P(b, V)$

$P(X_1, Y_1) \leftarrow R(X_1), Q(Y_1)$

$\eta_2 = \{X_1/b, V/Y_1\}$

$?R(b), Q(Y_1)$

$Q(c) \leftarrow$

$\eta_3 = \{Y_1/c\}$

$?R(b)$

$R(b) \leftarrow$

$\eta_4 = \varepsilon$

$?\square$

$\theta = \theta_1\theta_2\theta_3\theta_4|_{\{U,V\}} = \{U/b, V/c\}$

$\eta = \eta_1\eta_2\eta_3\eta_4|_{\{U,V\}} = \{U/b, V/c\}$

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Как видите, мы получаем одинаковые вычисленные ответы в обоих вычислениях независимо от порядка выбора подцелей в целевых утверждениях.

Это случайность или так бывает всегда?

С точки зрения операционной семантики программ, запрос — это список задач, которые нужно решить.

- Для решения запроса нужно решить **все** подцели запроса. Значит, результат вычисления не зависит от того порядка, в котором решаются задачи (выбираются подцели).
- Решение одной задачи может помочь решить другую. Значит, правильно выбранный порядок решения задач существенно влияет на эффективность вычисления запроса.

Покажем, что верно первое предположение.

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Определение.

Отображение R, которое сопоставляет каждому непустому запросу $G : ?C_1, C_2, \ldots, C_m$ одну из подцелей $C_i = R(G)$ в этом запросе, называется **правилом выбора подцелей** .

Для заданного правила выбора подцелей R вычисление запроса G к логической программе $\mathcal{P}$ называется **R-вычислением** , если на каждом шаге вычисления очередная подцель в запросе выбирается по правилу R.

Ответ, полученный в результате успешного R-вычисления, называется **R-вычисленным** .

Возникает вопрос:

При каком правиле выбора подцелей R каждый вычисленный ответ оказывется R-вычисленным?

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Для каждого программного утверждения $D: \ A_0 \leftarrow A_1, A_2, \ldots, A_n$ условимся использовать запись D^+ для обозначения атома A_0, а запись D^- — для обозначения списка атомов $A_1, A_2, \ldots, A_n$. В частности, если D — это факт $A_0 \leftarrow$, то D^- — это пустой список.

Выбор обозначений обусловлен тем, что утверждение $D: \ A_0 \leftarrow A_1, A_2, \ldots, A_n$ соответствует дизъюнкту

$$\overbrace{A_0}^{\text{положительная литера}} \qquad \vee \underbrace{\neg A_1 \vee \neg A_2 \vee \cdots \vee \neg A_n}_{\text{отрицательные литеры}}$$

ПОЛНОТА ОПЕРАЦИОННОЙ СЕМАНТИКИ

Переключательная лемма.

Предположим, что запрос $G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$ к хорновской логической программе $\mathcal{P}$ имеет вычисление

$G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$

D_1

$\theta_1 \in HOV(C_i, D_1^+)$

$G_1' :? (C_1, \ldots, D_1^-, \ldots, C_j, \ldots, C_m)\theta_1$

D_2

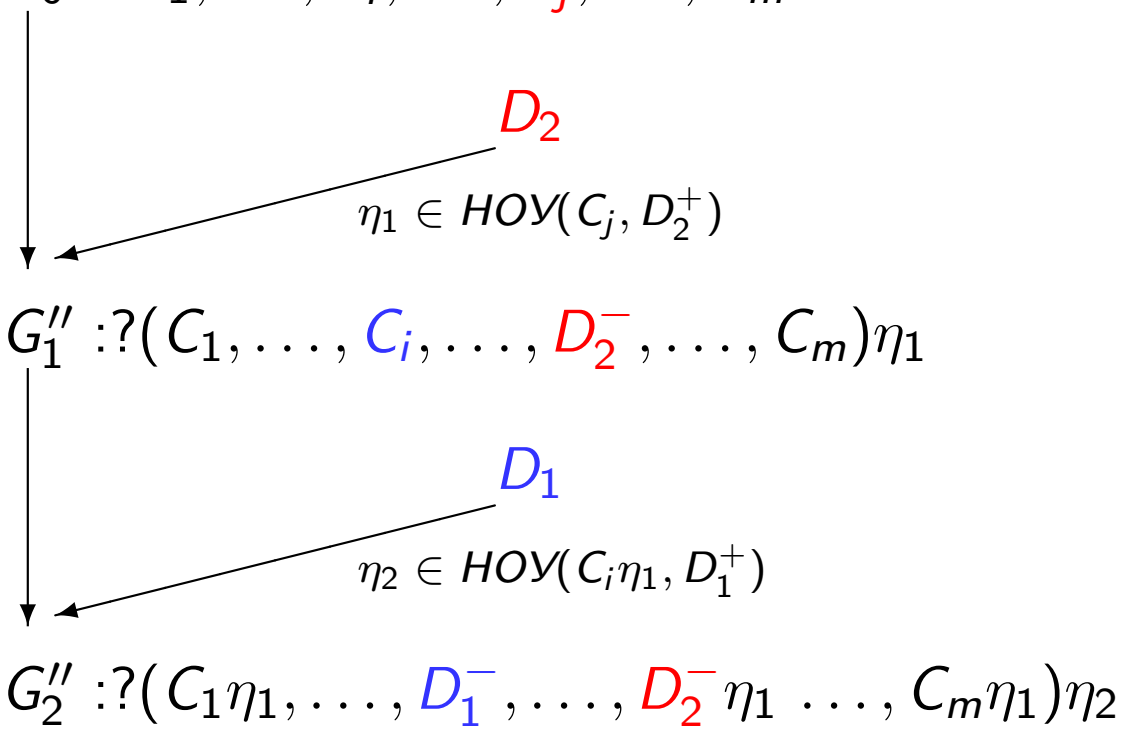
$\theta_2 \in HOV(C_j\theta_1, D_2^+)$

$G_2' :? (C_1\theta_1, \ldots, D_1^-\theta_1, \ldots, D_2^-, \ldots, C_m\theta_1)\theta_2$

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Переключательная лемма.

Тогда запрос G_0 к программе $\mathcal{P}$ также имеет вычисление



и при этом запросы G_2' и G_2'' являются вариантами друг друга, т. е. $\theta_1\theta_2\rho' = \eta_1\eta_2$ и $\eta_1\eta_2\rho'' = \theta_1\theta_2$ для некоторых $\rho', \rho'' \in Subst$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

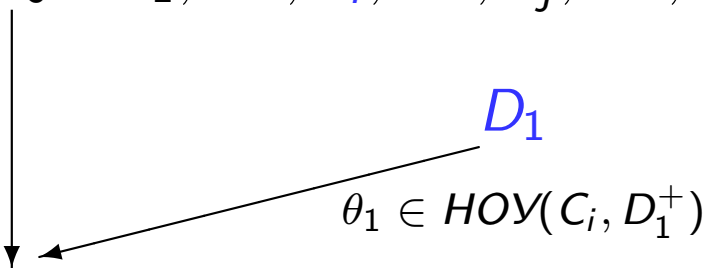
Переключательная лемма говорит о том, что при изменении порядка выбора подцелей результат вычисления сохраняется (с точностью до переименования переменных).

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

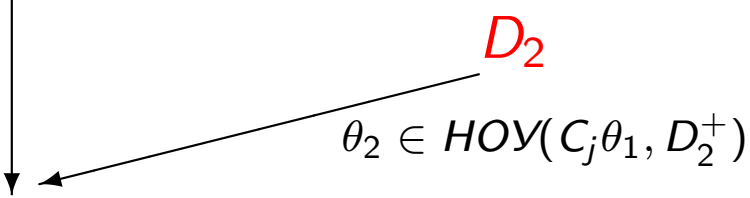
Доказательство (переключательной леммы).

Рассмотрим первое вычисление

$G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$



$G_1' :? (C_1, \ldots, D_1^-, \ldots, C_j, \ldots, C_m)\theta_1$

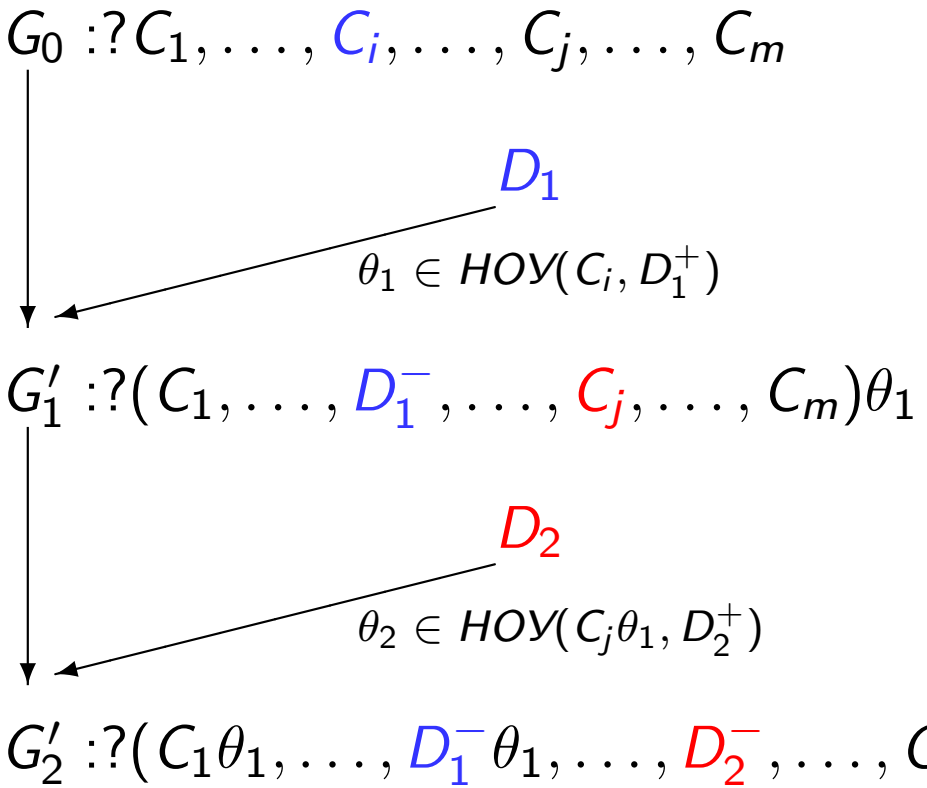


$G_2' :? (C_1\theta_1, \ldots, D_1^-\theta_1, \ldots, D_2^-, \ldots, C_m\theta_1)\theta_2$

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление



Согласно определению SLD-резолютивного вычисления $Dom_{\theta_1} \cap Var_{D_2} = \emptyset$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление

$G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$

D_1

$\theta_1 \in HOY(C_i, D_1^+)$

$G_1' :? (C_1, \ldots, D_1^-, \ldots, C_j, \ldots, C_m)\theta_1$

D_2

$\theta_2 \in HOY(C_j\theta_1, D_2^+)$

$G_2' :? (C_1\theta_1, \ldots, D_1^-\theta_1, \ldots, D_2^-, \ldots, C_m\theta_1)\theta_2$

Согласно определению SLD-резолютивного вычисления $Dom_{\theta_1} \cap Var_{D_2} = \emptyset$.

Поэтому $D_2^+\theta_2 = D_2^+\theta_1\theta_2$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление

$G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$

D_1

$\theta_1 \in HOY(C_i, D_1^+)$

$G_1' :? (C_1, \ldots, D_1^-, \ldots, C_j, \ldots, C_m)\theta_1$

D_2

$\theta_2 \in HOY(C_j\theta_1, D_2^+)$

$G_2' :? (C_1\theta_1, \ldots, D_1^-\theta_1, \ldots, D_2^-, \ldots, C_m\theta_1)\theta_2$

Согласно определению SLD-резолютивного вычисления $Dom_{\theta_1} \cap Var_{D_2} = \emptyset$.

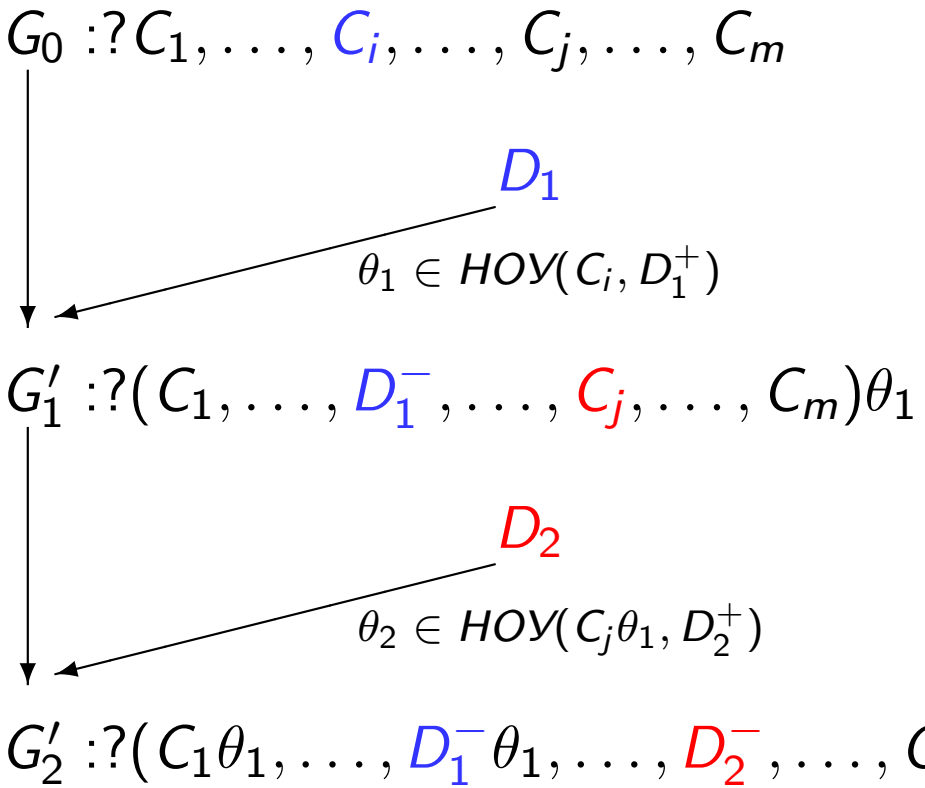
Поэтому $D_2^+\theta_2 = D_2^+\theta_1\theta_2$.

Следовательно,
$C_j\theta_1\theta_2 = D_2^+\theta_2 = D_2^+\theta_1\theta_2$,
т. е. C_j и D_2^+ унифицируемы.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление

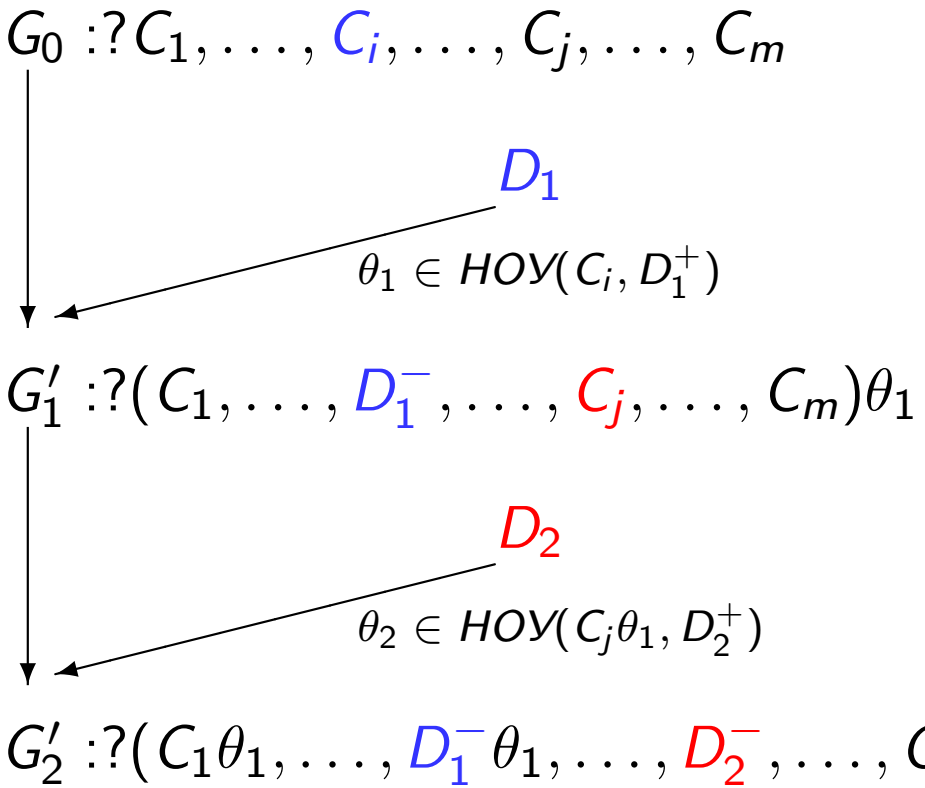


А раз C_j и D_2^+ унифицируемы, существует $\eta_1 \in HOY(C_j, D_2^+)$, и при этом $\theta_1\theta_2 = \eta_1\lambda$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление



Далее, заметим,
что $C_i\theta_1\theta_2 = D_1^+\theta_1\theta_2$,
и поэтому $C_i\eta_1\lambda = D_1^+\eta_1\lambda$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление

$G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$

D_1

$\theta_1 \in HOY(C_i, D_1^+)$

$G_1' :? (C_1, \ldots, D_1^-, \ldots, C_j, \ldots, C_m)\theta_1$

D_2

$\theta_2 \in HOY(C_j\theta_1, D_2^+)$

$G_2' :? (C_1\theta_1, \ldots, D_1^-\theta_1, \ldots, D_2^-, \ldots, C_m\theta_1)\theta_2$

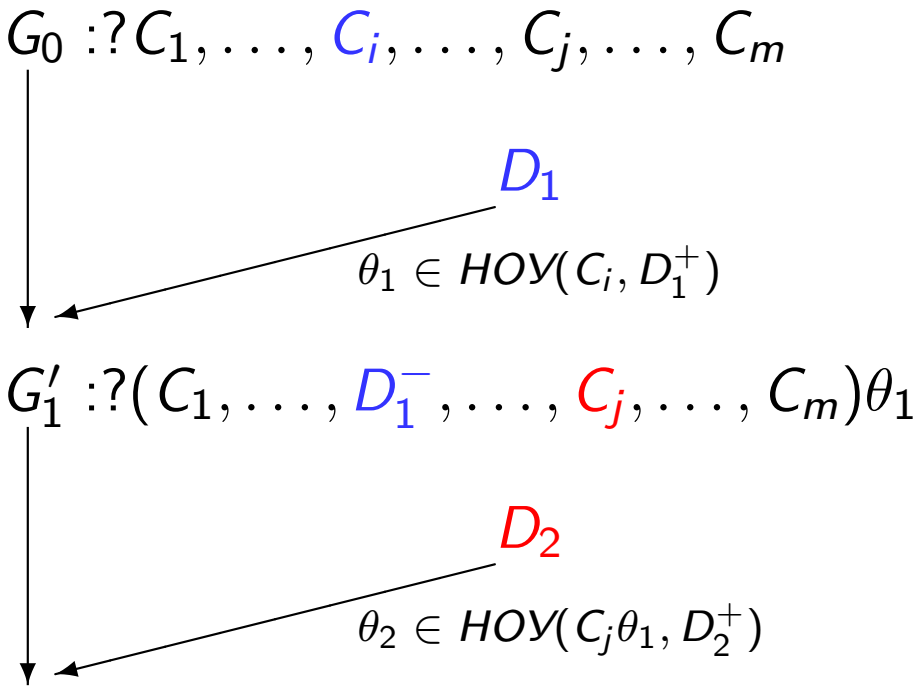
Далее, заметим,
что $C_i\theta_1\theta_2 = D_1^+\theta_1\theta_2$,
и поэтому $C_i\eta_1\lambda = D_1^+\eta_1\lambda$.

Т. к. $Dom_{\eta_1} \cap Var_{D_1} = \emptyset$,
верно $D_1^+\eta_1 = D_1^+$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление



Далее, заметим,
что $C_i\theta_1\theta_2 = D_1^+\theta_1\theta_2$,
и поэтому $C_i\eta_1\lambda = D_1^+\eta_1\lambda$.

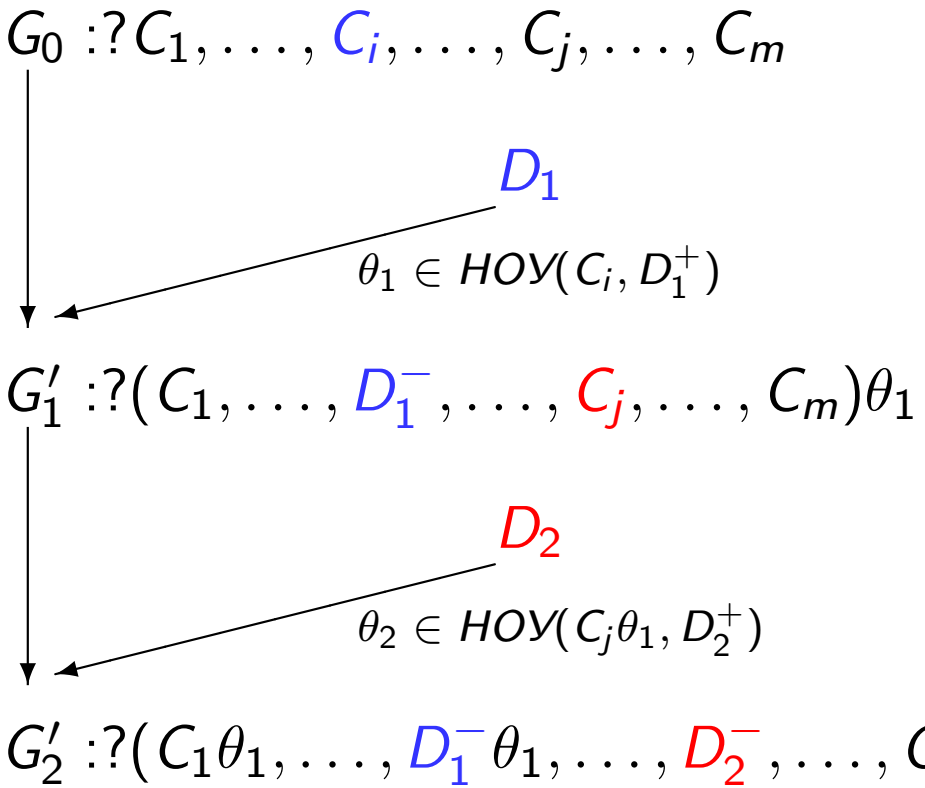
Т. к. $Dom_{\eta_1} \cap Var_{D_1} = \emptyset$,
верно $D_1^+\eta_1 = D_1^+$.

Значит, $C_i\eta_1\lambda = D_1^+\lambda$,
т. е. $C_i\eta_1$ и D_1^+ унифицируемы.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление

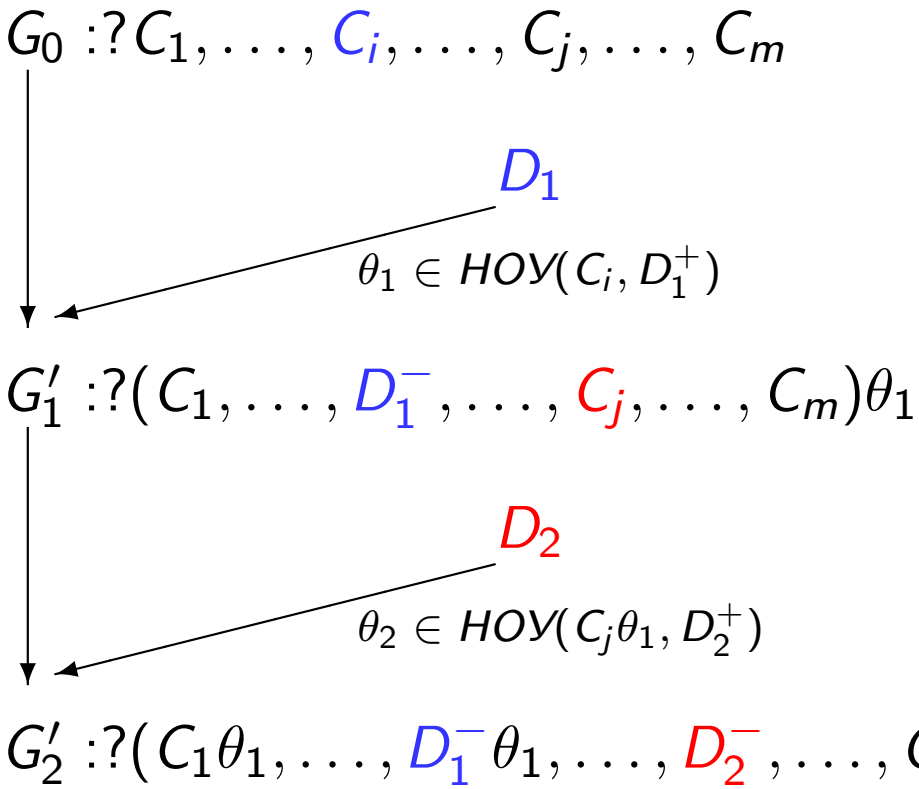


А раз $C_i\eta_1$ и D_1^+ унифицируемы, существует $\eta_2 \in HOV(C_i\eta_1, D_1^+)$, и при этом $\lambda = \eta_2\rho'$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление



Итак, получаем

$\eta_1 \in HOY(C_j, D_2^+)$,

$\eta_2 \in HOY(C_i\eta_1, D_1^+)$,

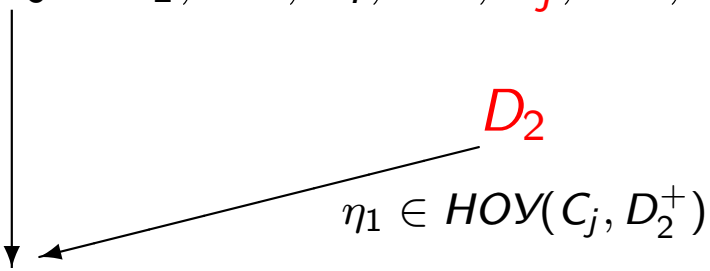
$\theta_1\theta_2 = \eta_1\lambda = \eta_1\eta_2\rho'$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

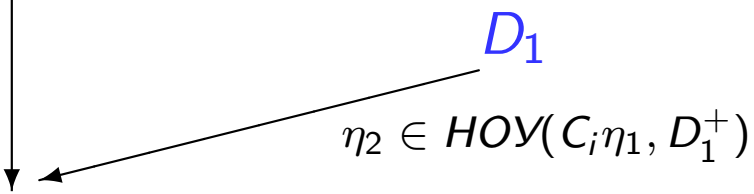
Доказательство (переключательной леммы).

Получаем другое вычисление

$G_0 :? C_1, \ldots, C_i, \ldots, C_j, \ldots, C_m$



$G_1'' :? (C_1, \ldots, C_i, \ldots, D_2^-, \ldots, C_m)\eta_1$



$G_2'' :? (C_1\eta_1, \ldots, D_1^-, \ldots, D_2^-\eta_1 \ \ldots, C_m\eta_1)\eta_2$

Итак, получаем

$\eta_1 \in HOY(C_j, D_2^+)$,

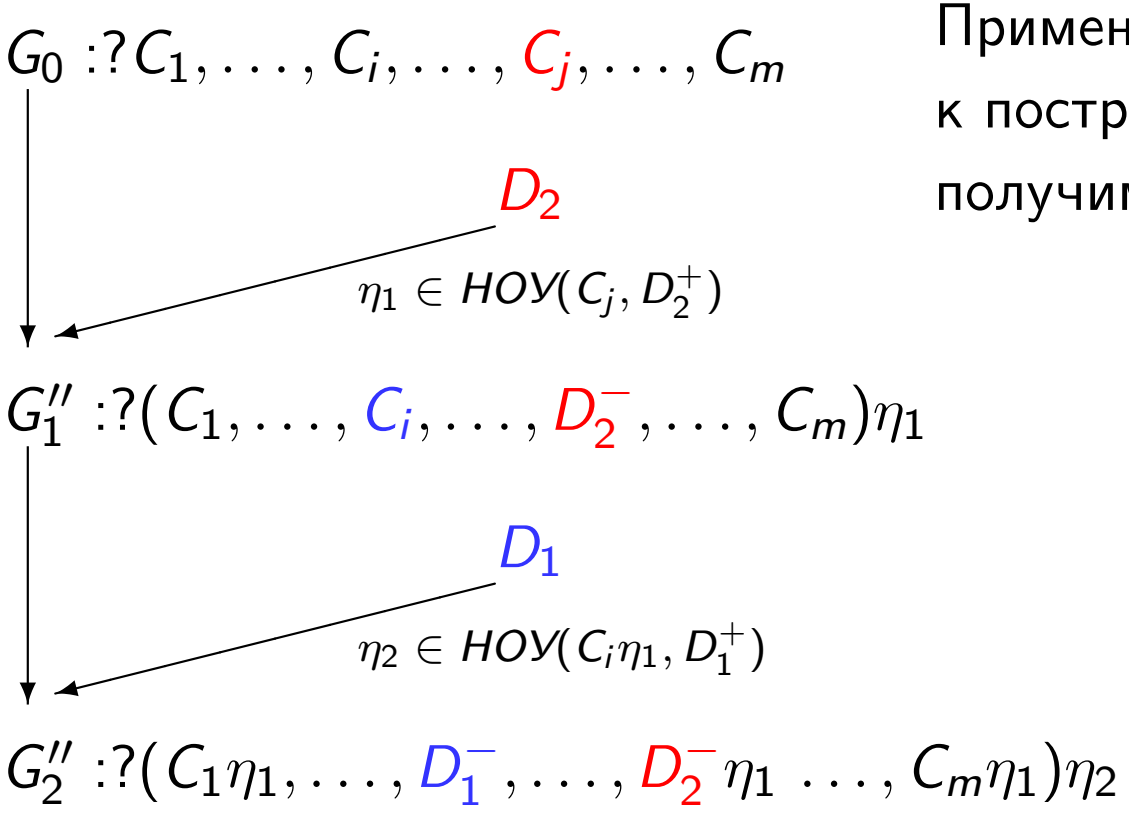
$\eta_2 \in HOY(C_i\eta_1, D_1^+)$,

$\theta_1\theta_2 = \eta_1\lambda = \eta_1\eta_2\rho'$.

Значит, запрос G_0 имеет и другое вычисление.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

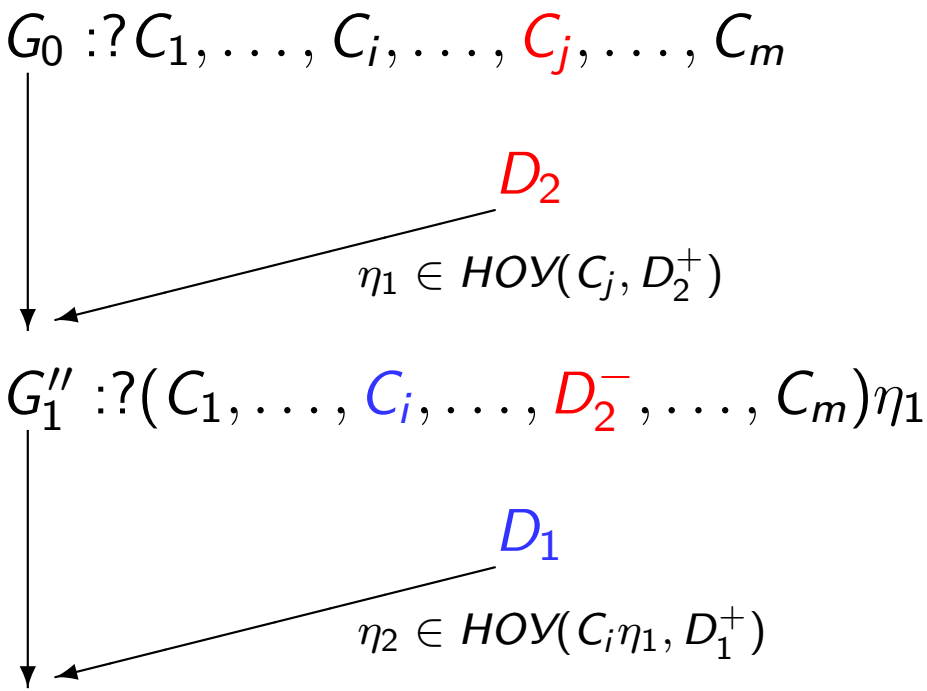
Доказательство (переключательной леммы).



Применяя те же рассуждения к построенному вычислению получим $\eta_1\eta_2 = \theta_1\theta_2\rho''$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).



Применяя те же рассуждения к построенному вычислению получим $\eta_1\eta_2 = \theta_1\theta_2\rho''$.

Равенства $\theta_1\theta_2 = \eta_1\eta_2\rho'$, $\eta_1\eta_2 = \theta_1\theta_2\rho''$ означают, что подстановки $\eta_1\eta_2$ и $\theta_1\theta_2$, а также запросы G_2' и G_2'' являются вариантами друг друга. $\square$

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Теорема сильной полноты

Каково бы ни было правило выбора подцелей R, если θ — правильный ответ на запрос G_0 к хорновской логической программе $\mathcal{P}$, то существует такой R-**вычисленный ответ** η, что равенство

$$\theta = \eta\rho$$

выполняется для некоторой подстановки ρ.

Доказательство

По теореме полноты существуют такой вычисленный ответ η', что $\theta = \eta'\rho'$. Рассмотрим соответствующее этому ответу успешное вычисление запроса G

$$comp' = (D_1, \eta'_1, G_1), \ldots, (D_N, \eta'_N, \square),$$

в котором $\eta' = \eta'_1 \ldots \eta'_N$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство

Предположим, что $R(G_0) = C_i$. Поскольку $comp'$ — это успешное вычисление, существует k_i, что подцель C_i впервые выбирается на k_i-ом шаге вычисления $comp'$. Применяя последовательно k_i раз переключательную лемму, можно получить успешное вычисление

$$comp'' = (D_{i_k}, \eta_1'', G_1''), (D_1, \eta_2'', G_1''), \ldots, (D_N, \eta_N'', \square),$$

в котором на первом шаге выбирается подцель $C_i = R(G_0)$, но при этом вычисленный результат $\eta'' = \eta_1'' \ldots \eta_N''$ — это вариант вычисленного ответа $\eta' = \eta_1' \ldots \eta_N'$, и значит $\theta = \eta'' \rho''$.

Повторяя этот трюк N раз, получим требуемое успешное R-вычисление.

Полное и строгое доказательство требует применения математической индукции. Провести самостоятельно. $\square$

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Теорема сильной полноты говорит о том. что правило выбора подцелей не играет существенной роли при вычислении ответа: любое правило выбора подцелей позволяет получить все вычисленные ответы.

Поэтому для единообразной организации вычислений логических программ всегда используется **стандартное правило выбора** : в каждом запросе всегда выбирается самая первая (левая) подцель.
Теперь займемся вопросом о том, какую роль играет выбор подходящих программных утверждений.

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

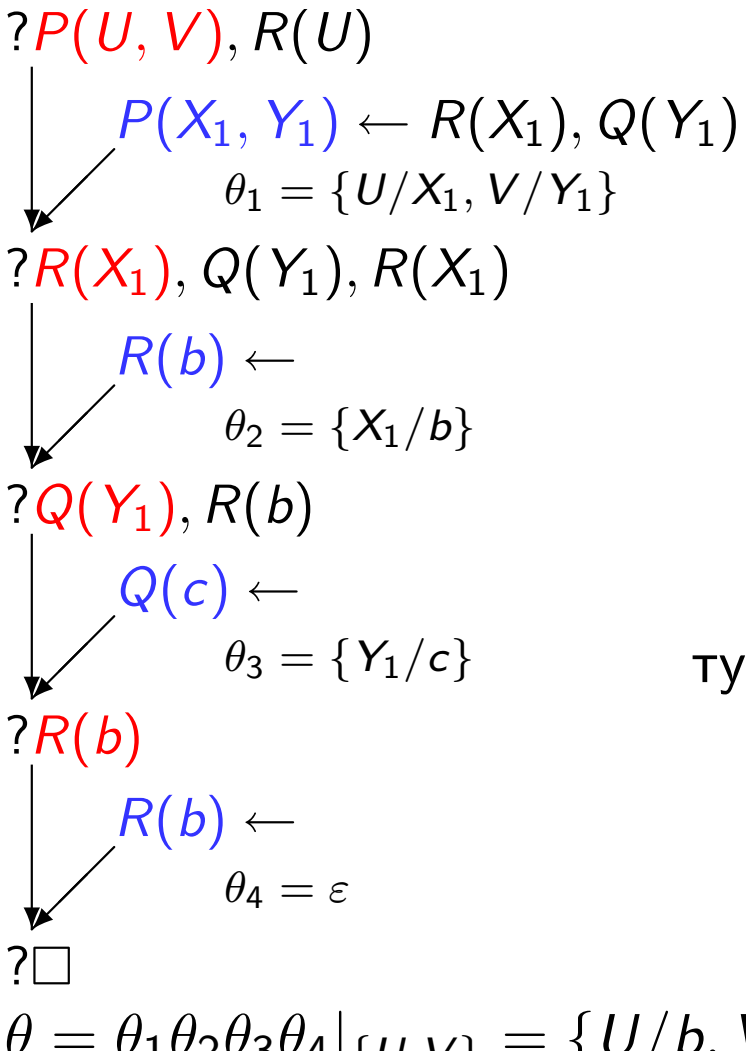
Обратимся снова к запросу $?P(U, V), R(U)$ к логической программе

$$\begin{array}{lll} \mathcal{P}: & P(X,Y) \leftarrow R(X), Q(Y); & (1) \\ & P(X,X) \leftarrow Q(X); & (2) \\ & R(b) \leftarrow; & (3) \\ & Q(c) \leftarrow; & (4) \end{array}$$

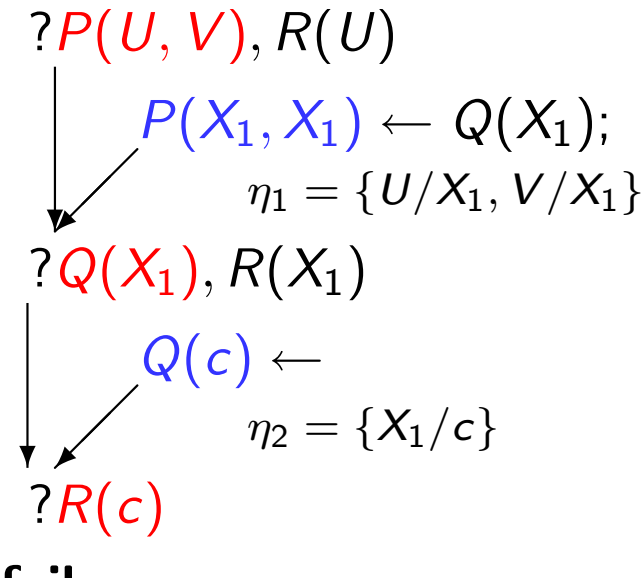
и будем применять стандартное правило выбора подцелей.

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Тогда возможны следующие два вычисления запроса $?P(U, V), R(U)$



$\theta = \theta_1\theta_2\theta_3\theta_4|_{\{U,V\}} = \{U/b, V/c\}$



failure

тупиковое вычисление

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Как видно из этого примера, выбор программных утверждений играет значительную роль.

Программное утверждение — это способ (рецепт) решения задачи (подцели). Ясно, что некоторые способы решения могут быть «хорошими», а некоторые — «плохими».

Таким образом, чтобы вычислить все ответы на запрос (или, что то же само, гарантировать вычисление хотя бы одного ответа), нужно уметь просматривать все варианты выбора программных утверждений. И нужно правильно организовать этот перебор.

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

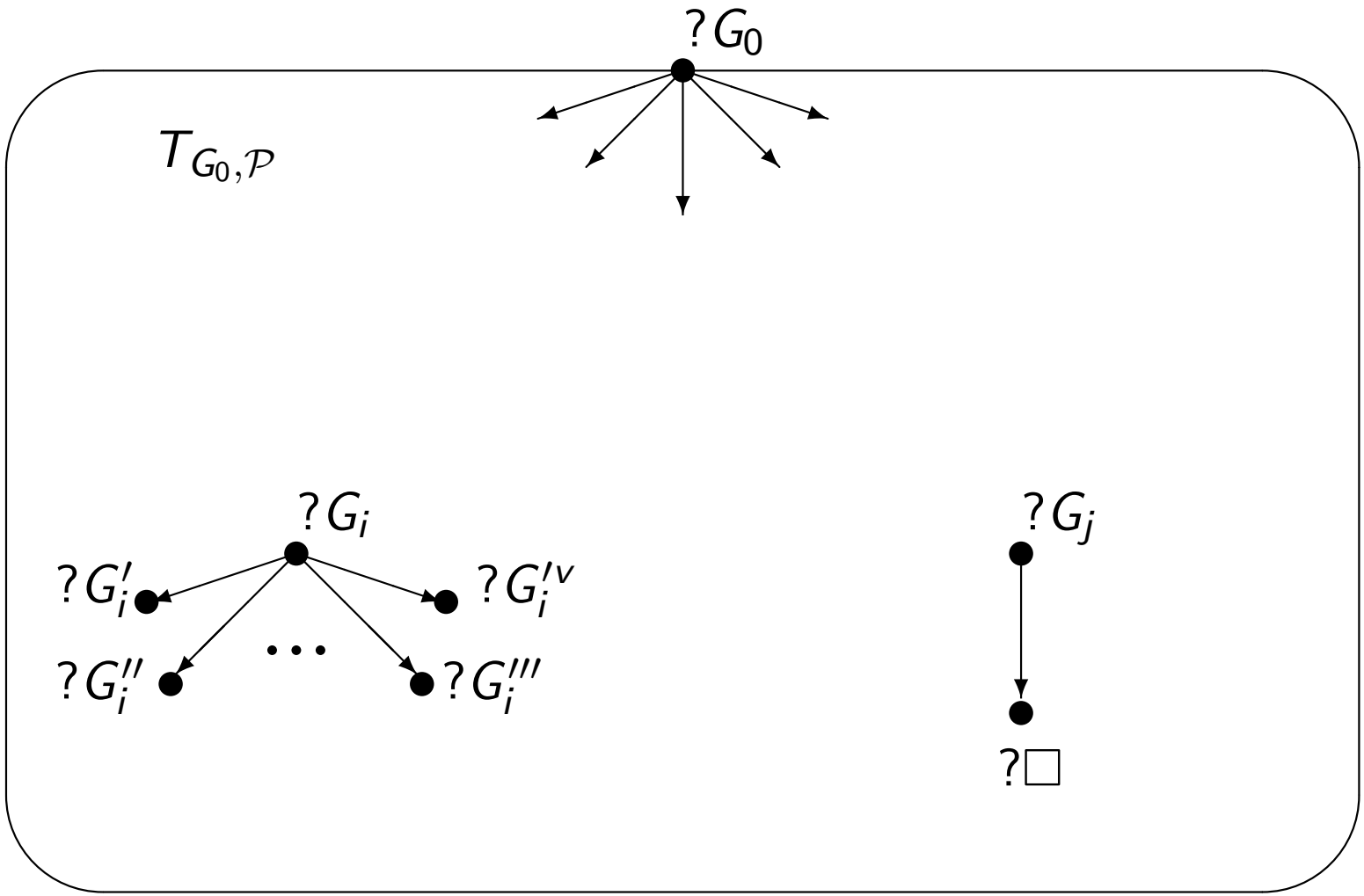
Определение

Деревом SLD-резолютивных вычислений запроса G_0 к логической программе $\mathcal{P}$ называется помеченное корневое дерево $T_{G_0,\mathcal{P}}$, удовлетворяющее следующим требованиям:

1. Корнем дерева является исходный запрос G_0;
2. Потомками каждой вершины G являются всевозможные SLD-резольвенты запроса G (при фиксированном стандартном правиле выбора подцелей);
3. Листовыми вершинами являются пустые запросы (завершающие успешные вычисления) и запросы, не имеющие SLD-резольвент (завершающие тупиковые вычисления).

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

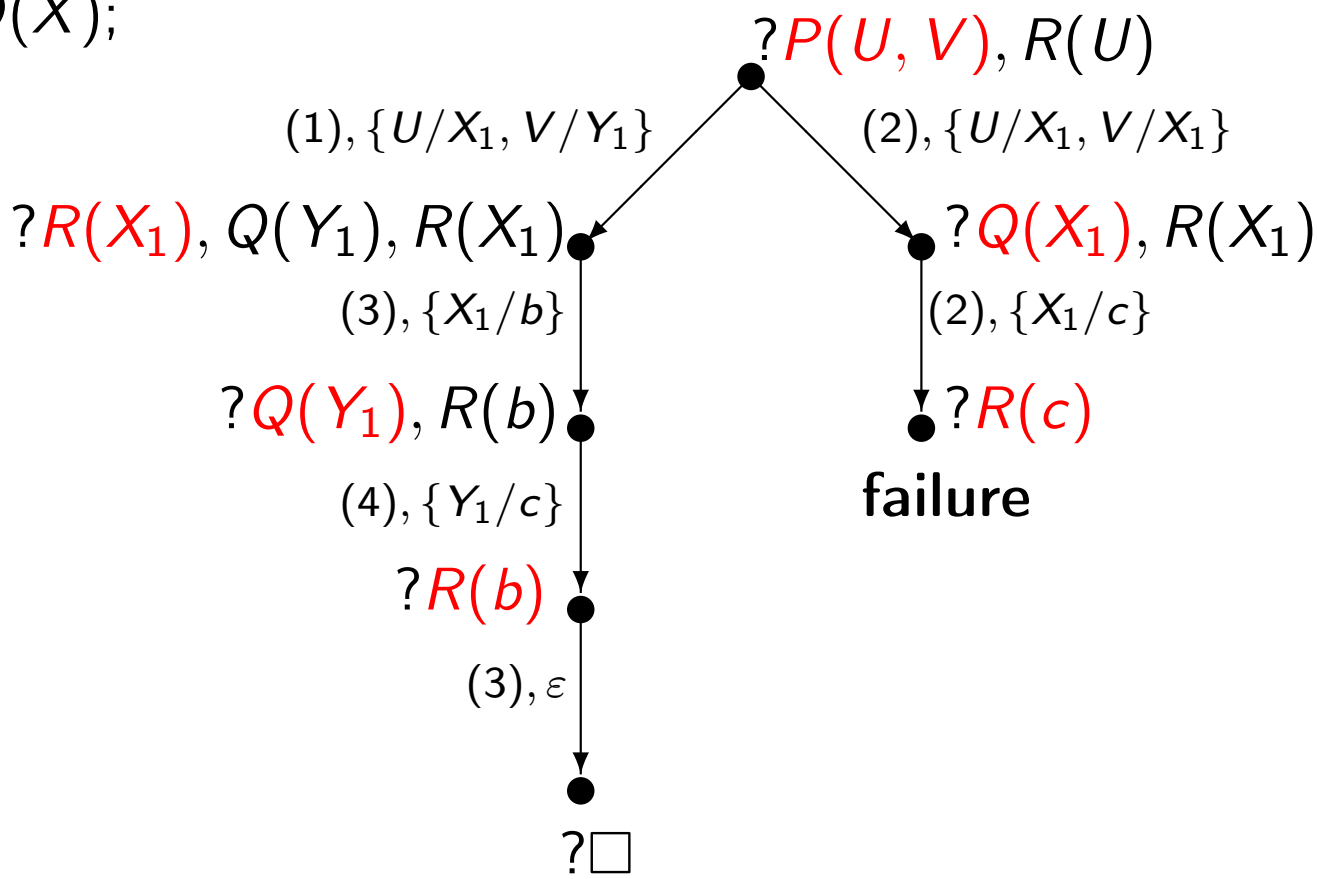
Иллюстрация



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример 1.

$\mathcal{P}$: $P(X, Y) \leftarrow R(X), Q(Y);$
$P(X, X) \leftarrow Q(X);$
$R(b) \leftarrow;$
$Q(c) \leftarrow;$



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

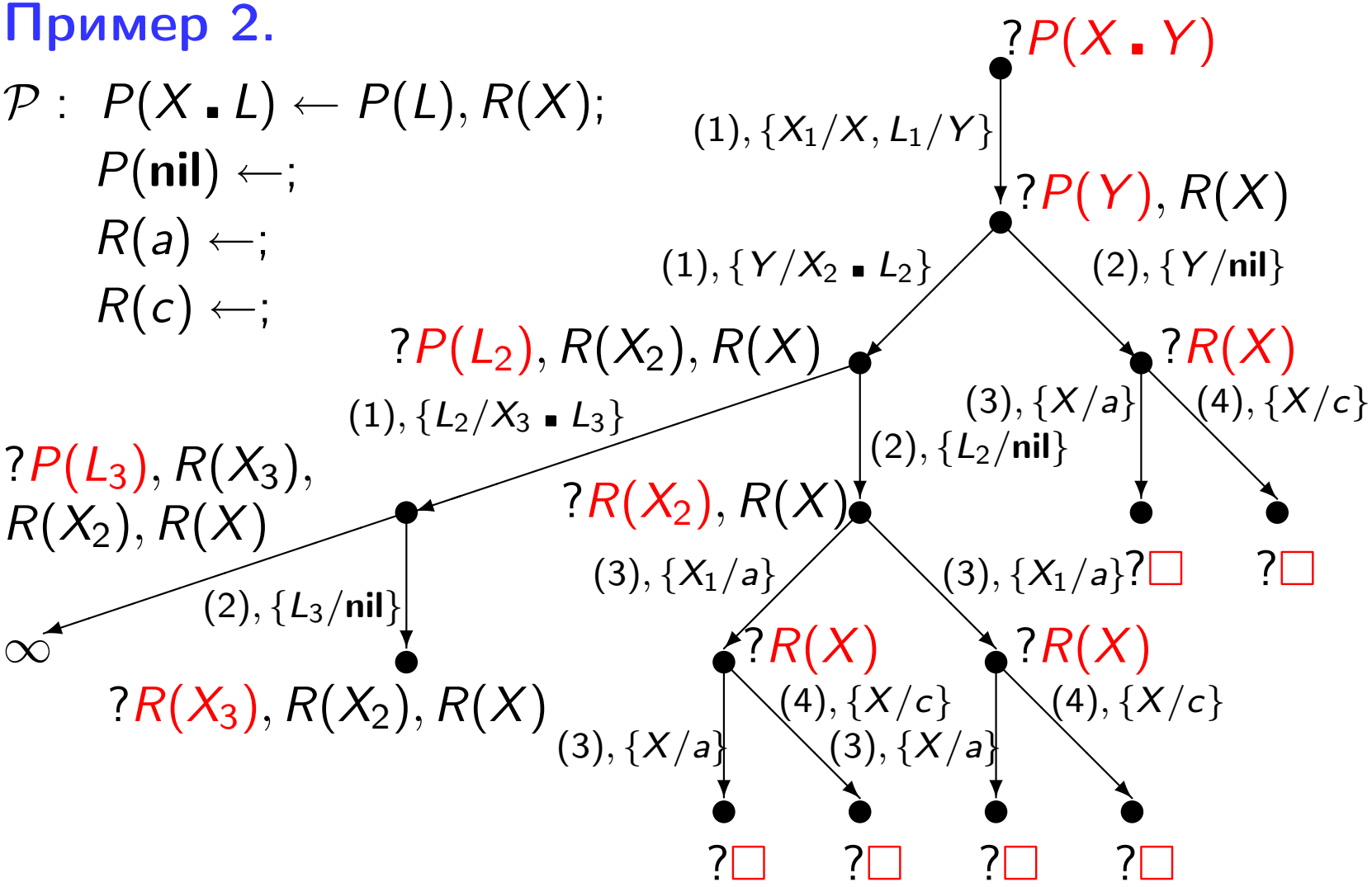
Пример 2.

$\mathcal{P}$: $P(X \cdot L) \leftarrow P(L), R(X);$

$P(\mathbf{nil}) \leftarrow;$

$R(a) \leftarrow;$

$R(c) \leftarrow;$



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Итак, деревья вычислений логических программ бывают разные — конечные и бесконечные, с конечным или бесконечным множеством ветвей, и т. п.

Каждая ветвь дерева $T_{G_0,\mathcal{P}}$ соответствует одному из возможных вычислений запроса G_0 к логической программе $\mathcal{P}$.

Некоторые из ветвей образуют успешное вычисление и дают ответ на запрос.

Возникает вопрос:

Как нам обнаружить ветви успешных вычислений в дереве SLD-резолютивных вычислений программы?

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Определение

Стратегией вычисления запросов к логическим программам называется алгоритм построения (обхода) дерева SLD-резолютивных вычислений $T_{G_0,\mathcal{P}}$ всякого запроса G_0 к произвольной логической программе $\mathcal{P}$

Стратегия вычислений называется **вычислительно полной**, если для любого запроса G_0 и любой логической программы $\mathcal{P}$ эта стратегия строит (обнаруживает) **все** успешные вычисления запроса G_0 к программы $\mathcal{P}$

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Фактически, стратегия вычисления — это одна стратегий обхода корневого дерева. Как известно, таких стратегий существует много, но среди них выделяются две наиболее характерные:

- **стратегия обхода в ширину**, при которой дерево строится (обходится) поярусно — вершина i-го не строится, до тех пор пока не будут построены все вершины $(i-1)$-го яруса;
- **стратегия обхода в глубину с возвратом**, при которой ветви дерева обходятся поочередно — очередная ветвь дерева не обохдится, до тех пор пока не будут пройдены все вершины текущей ветви.

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

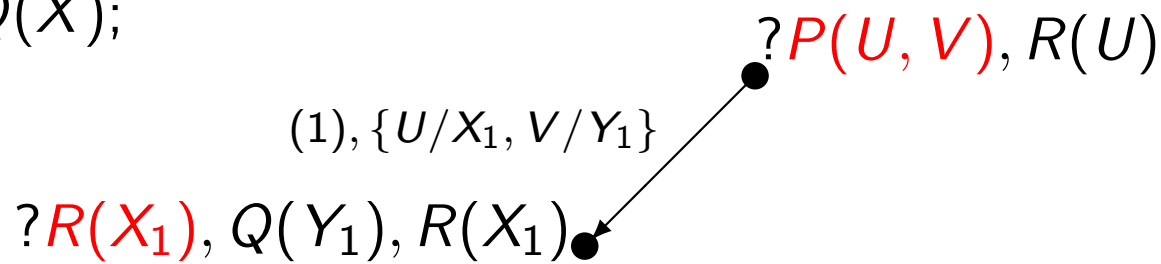
Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

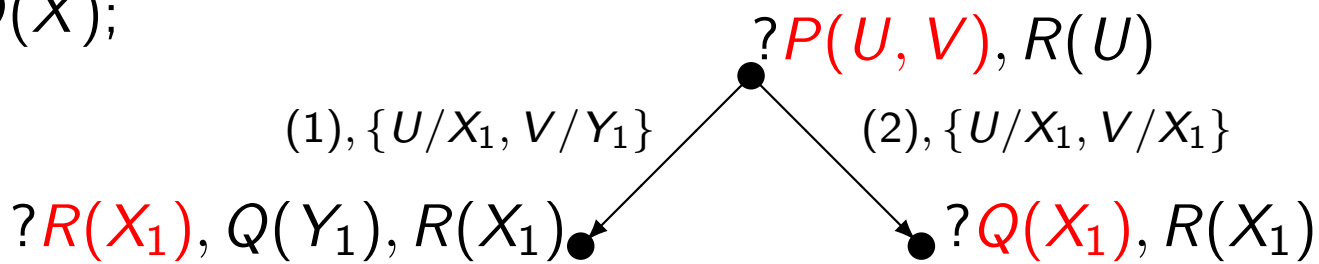
Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

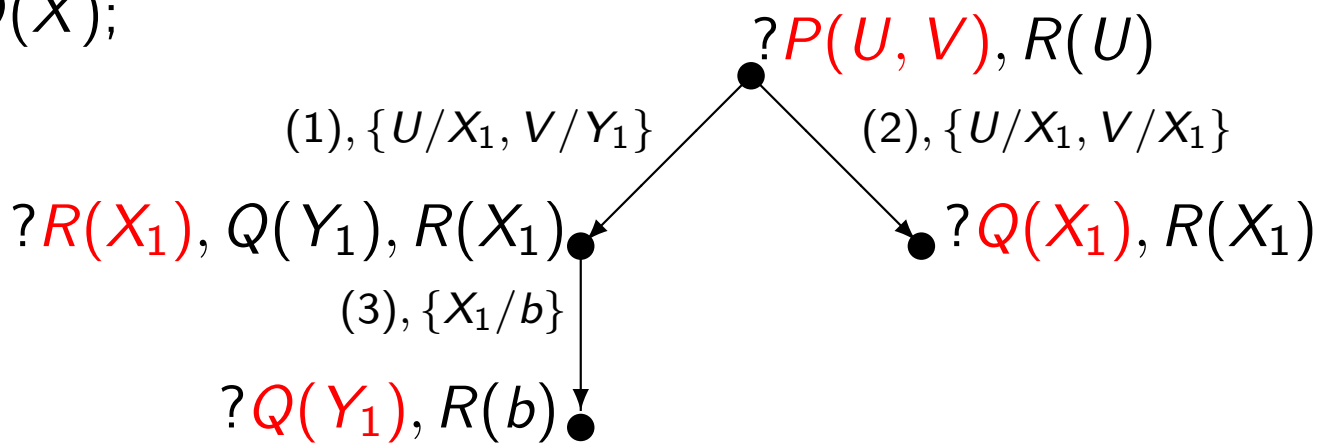
Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

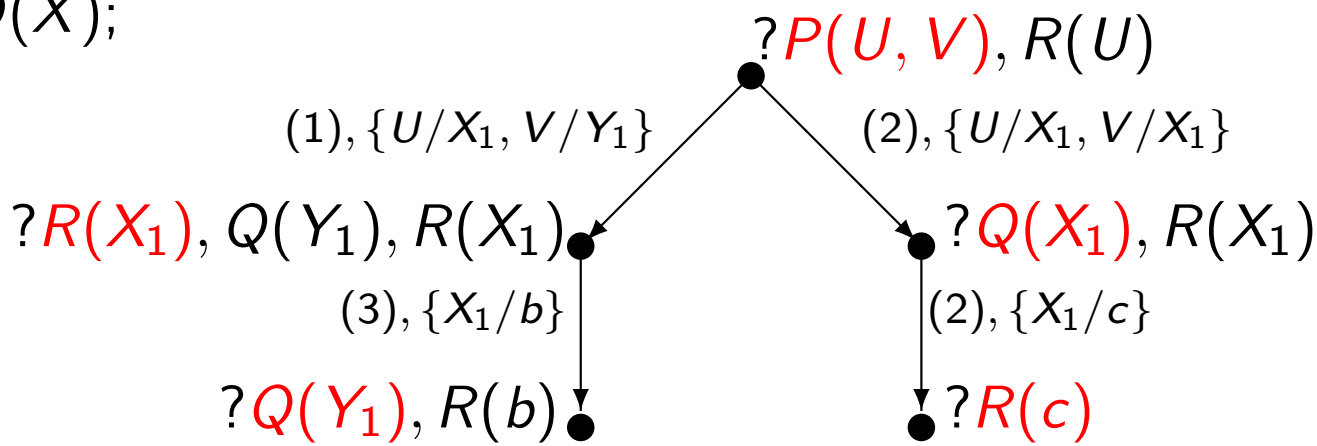
Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

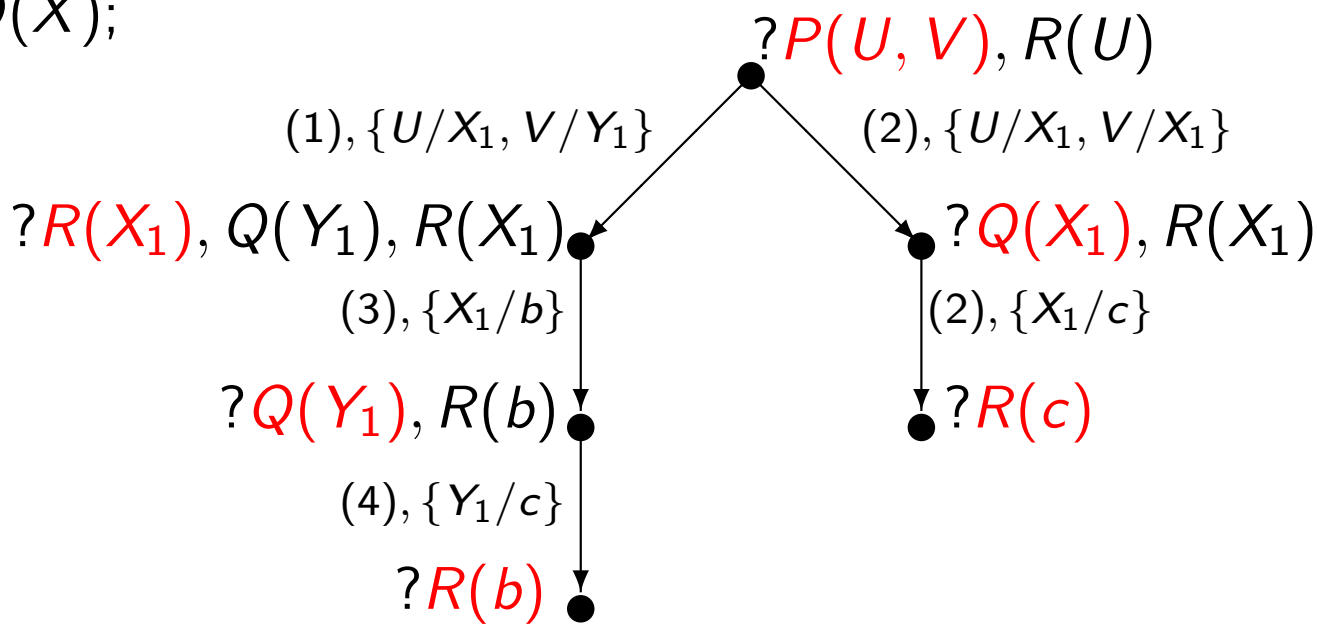
$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в ширину.

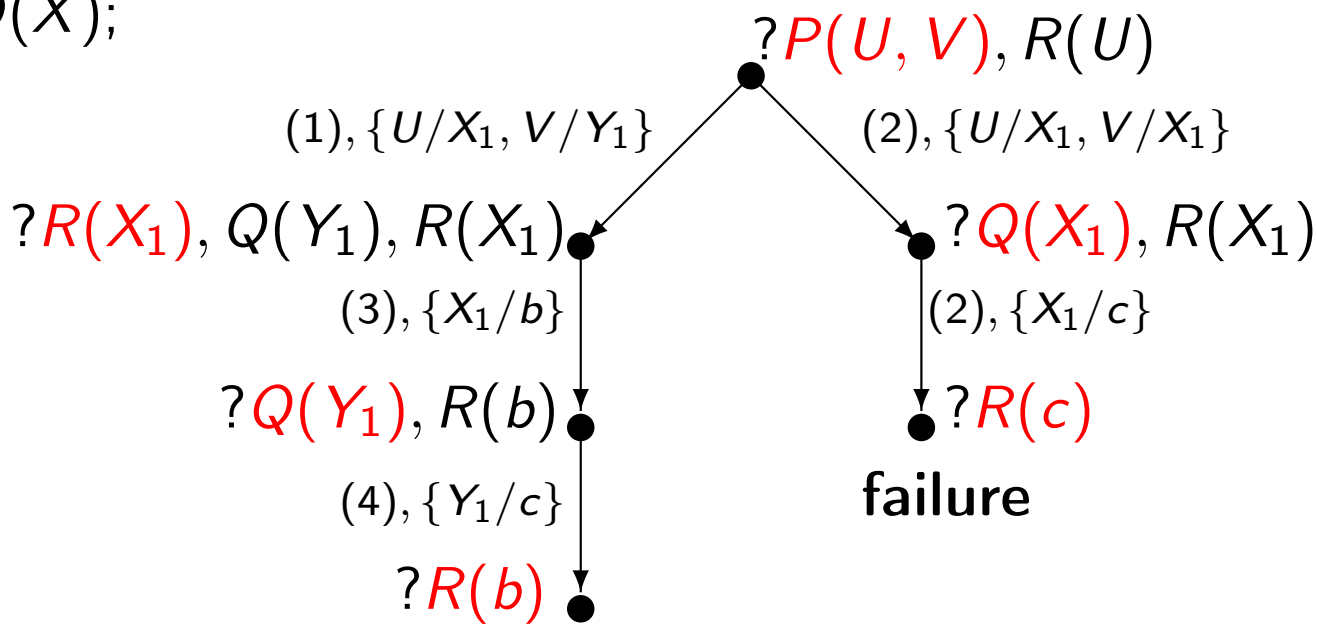
$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$
$P(X,X) \leftarrow Q(X);$
$R(b) \leftarrow;$
$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$
$P(X,X) \leftarrow Q(X);$
$R(b) \leftarrow;$
$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

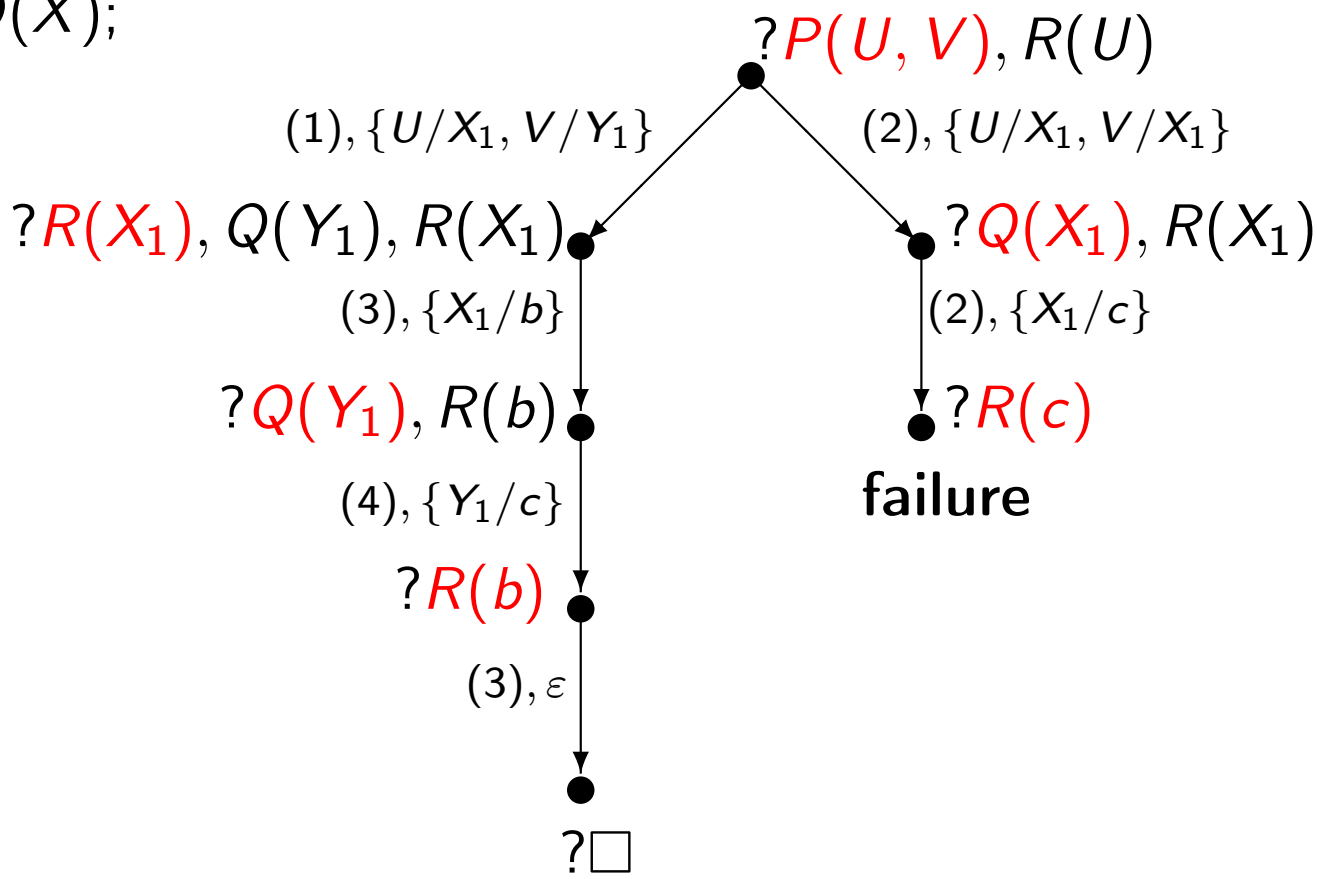
Пример обхода в ширину.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Стратегия обхода в ширину является вычислительно полной, поскольку

- каждый запрос имеет конечное число SLD-резольвент, и поэтому в каждом ярусе дерева SLD-резолютивных вычислений имеется конечное число вершин;
- каждое успешное вычисление завершается на некотором ярусе;
- и поэтому каждое успешное вычисление будет рано или поздно полностью построено.

Но строить интерпретатор логических программ на основе стратегии обхода в ширину **нецелесообразно**. При обходе дерева в ширину нужно обязательно хранить в памяти **все** вершины очередного яруса. Это требует большого расхода памяти. Например, в 100-м ярусе двоичного дерева содержится 2^{99} вершин. Вычислительных ресурсов всего земного шара не хватит, чтобы хранить информацию обо всех этих вершинах.

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Стратегия обхода в глубину с возвратом основана на следующих принципах:

1. все программные утверждения упорядочиваются;
2. на каждом шаге обхода из текущей вершины G осуществляется переход
   - либо в новую вершину-потомок G', которая является SLD-резольвентой запроса G и первого по порядку программного утверждения D, ранее не использованного для этой цели;
   - либо в ранее построенную родительскую вершину G'' (откат), если все программные утверждения уже были опробованы для построения SLD-резольвент запроса G.

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в глубину с возвратом.

$\mathcal{P}:$ $P(X, Y) \leftarrow R(X), Q(Y);$

$P(X, X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

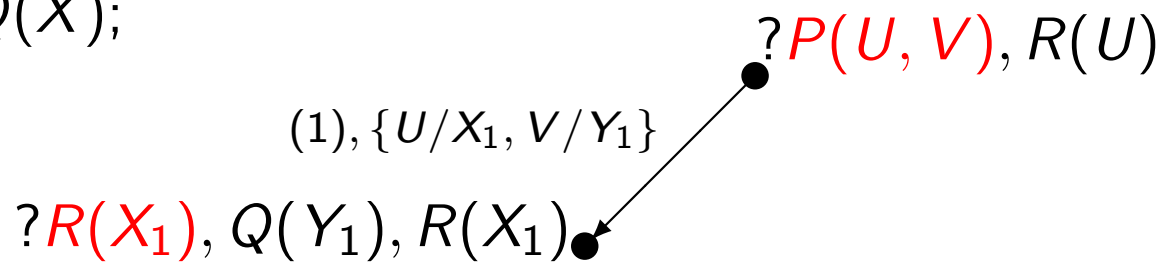
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

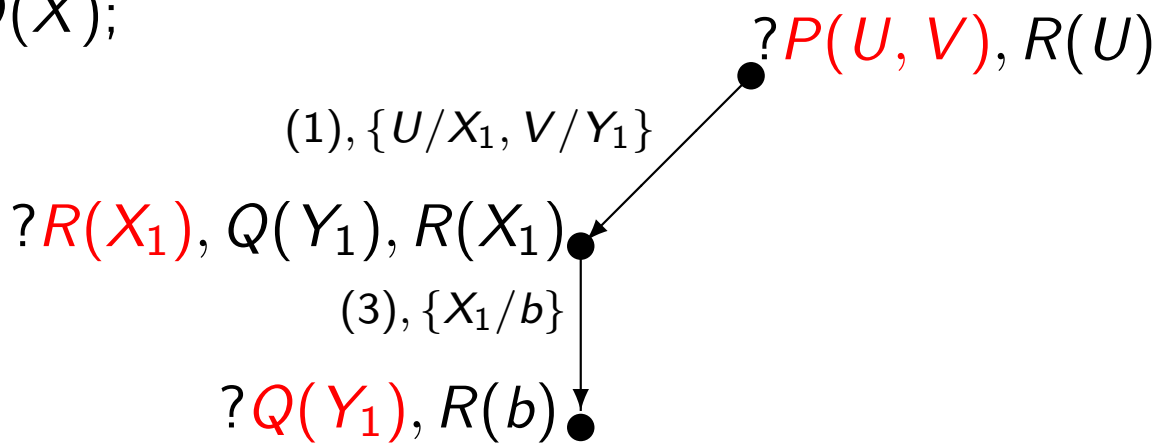
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

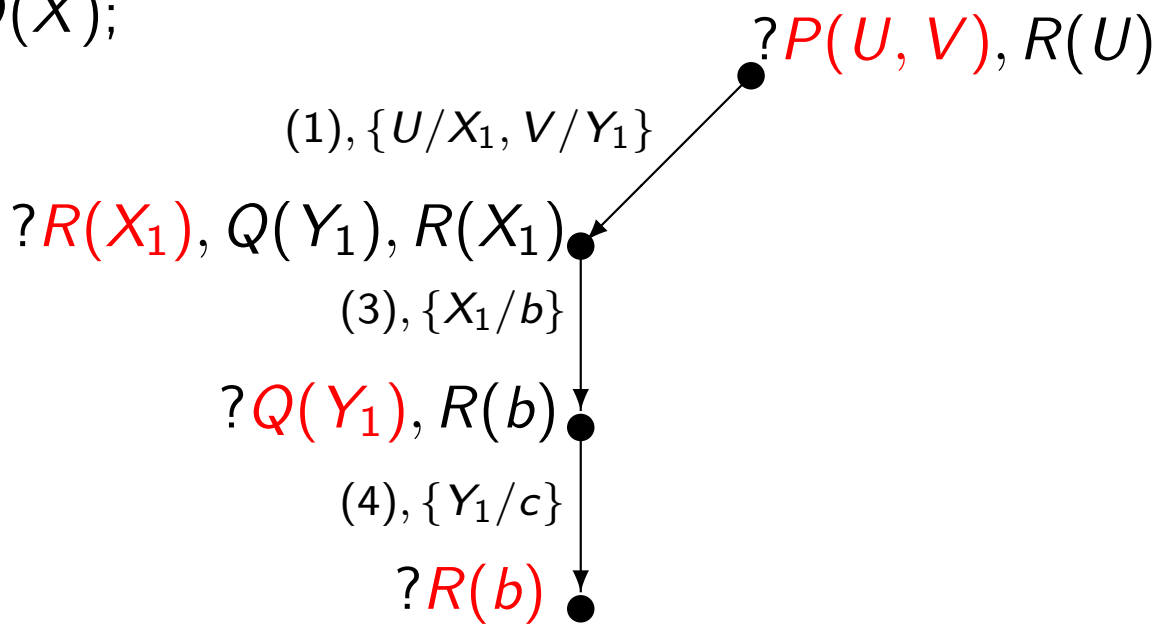
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

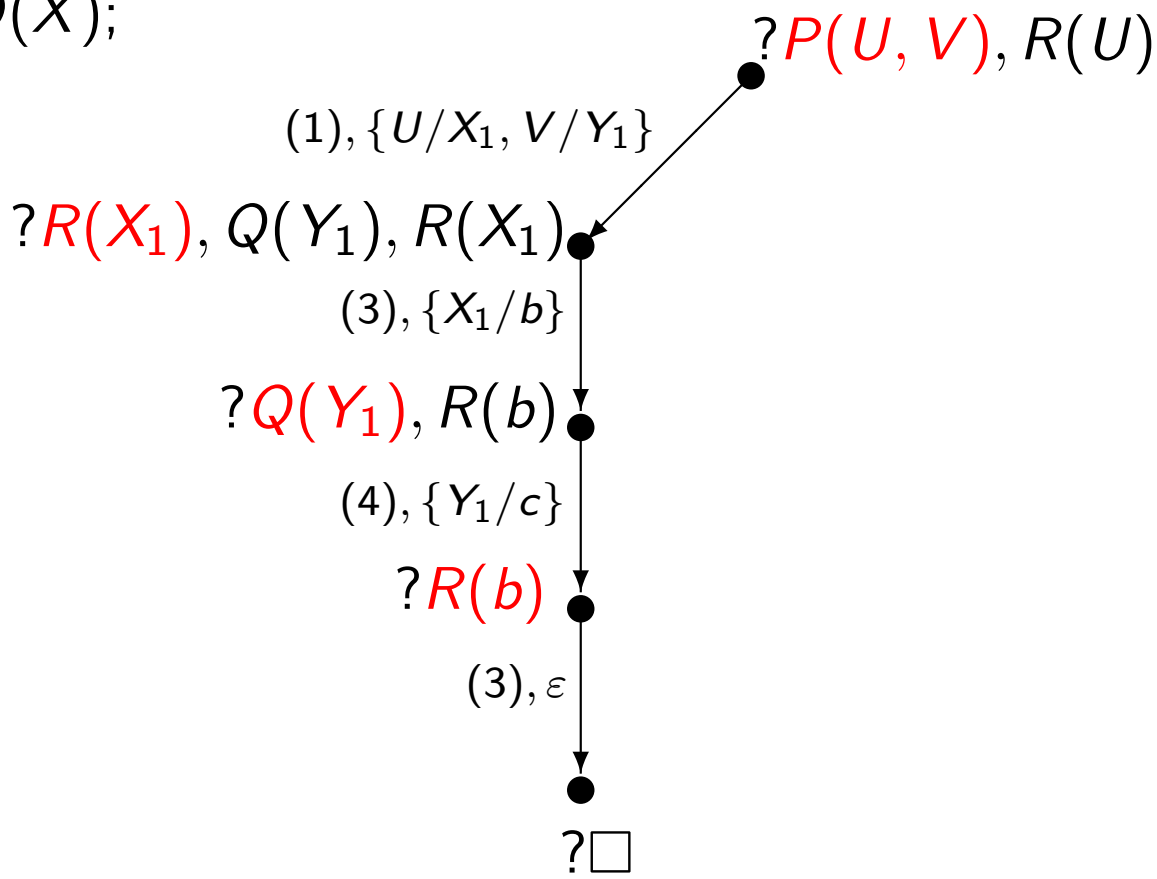
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

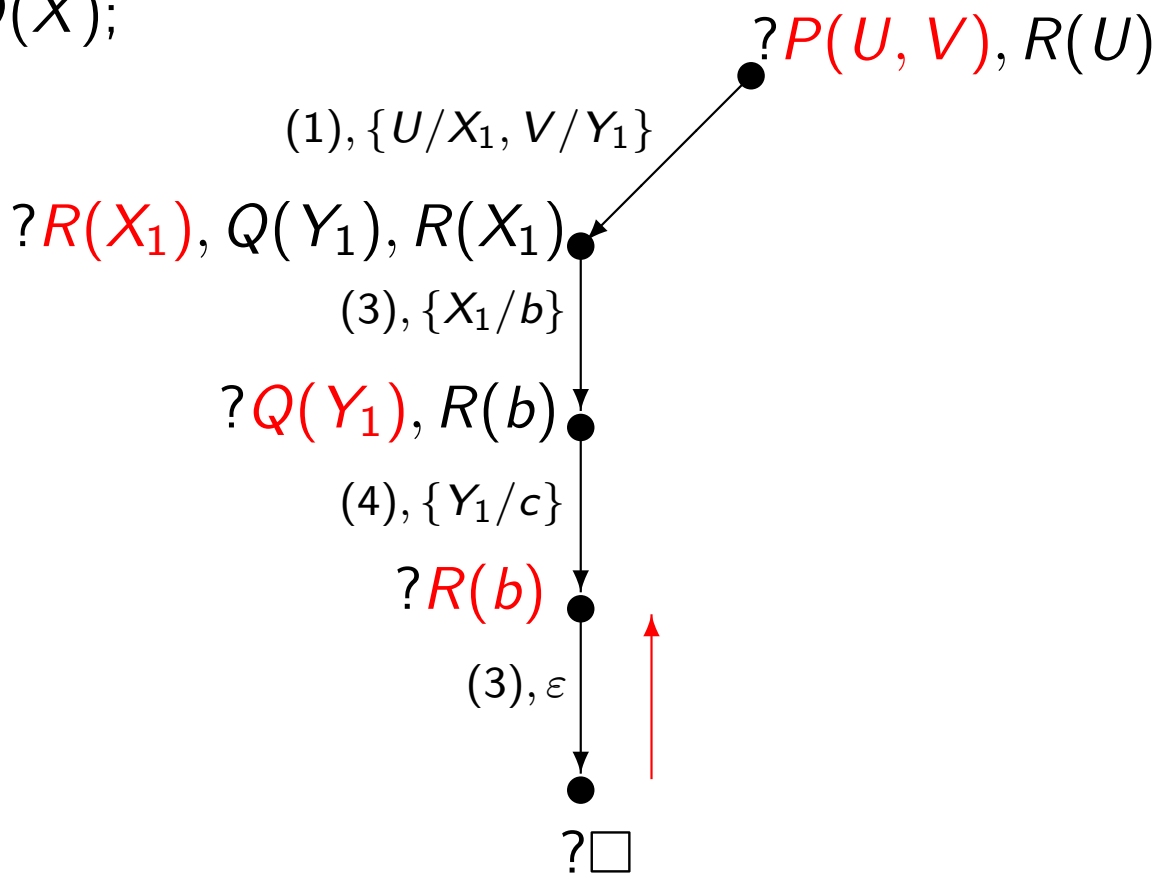
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

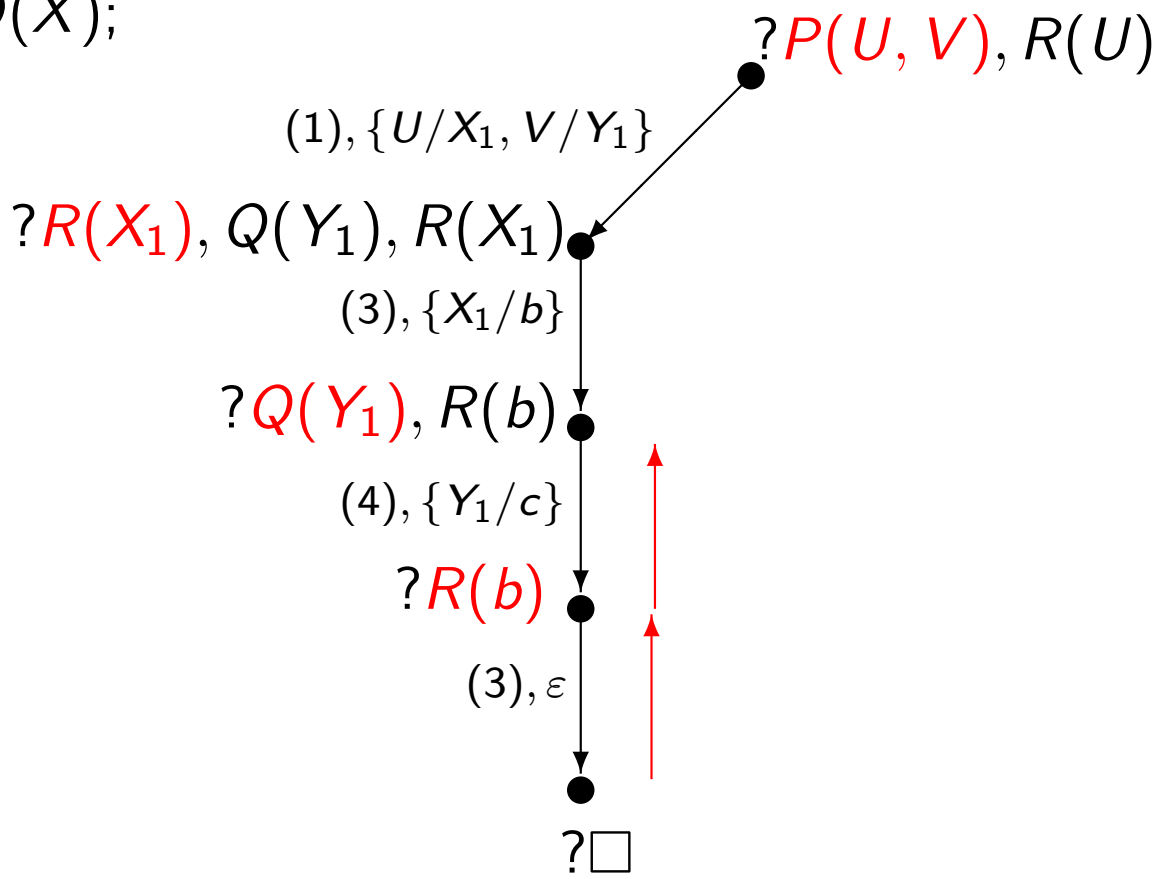
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

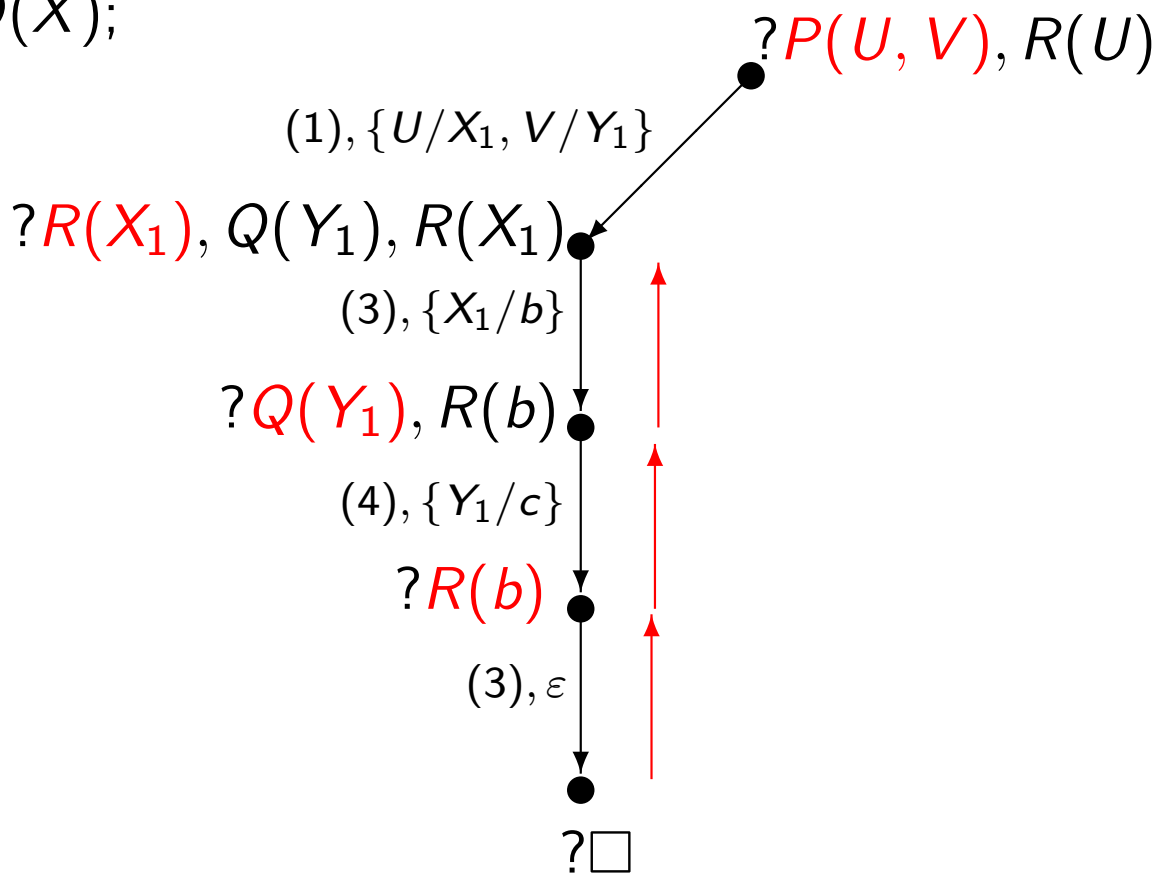
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

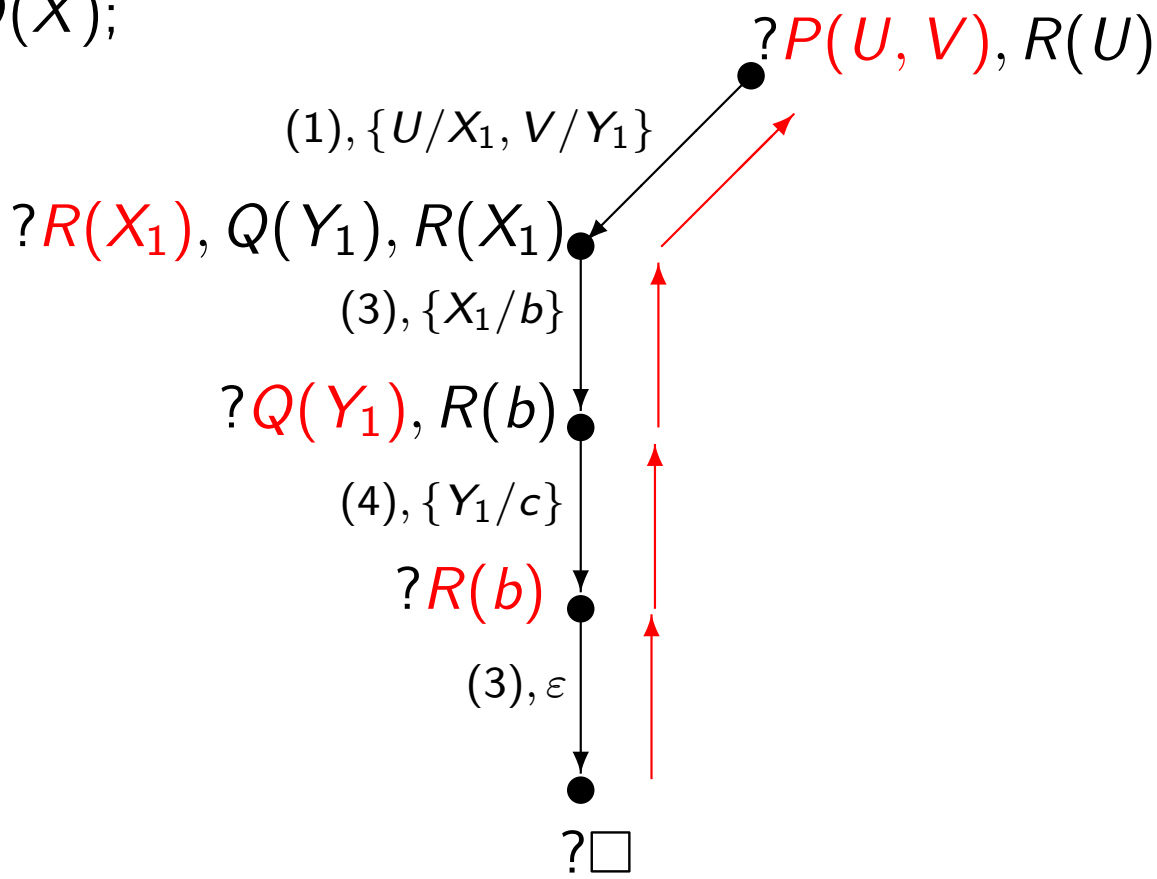
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

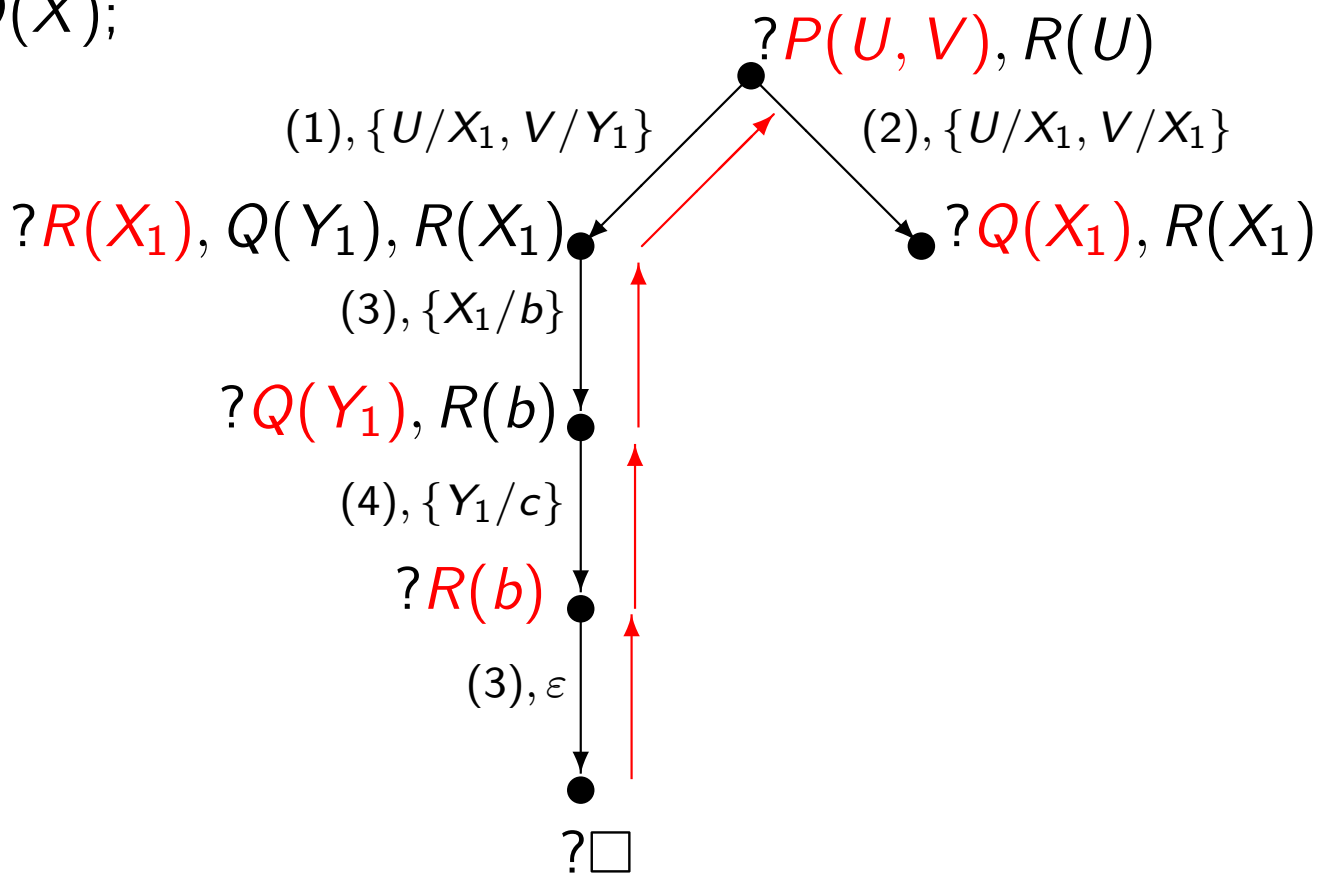
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

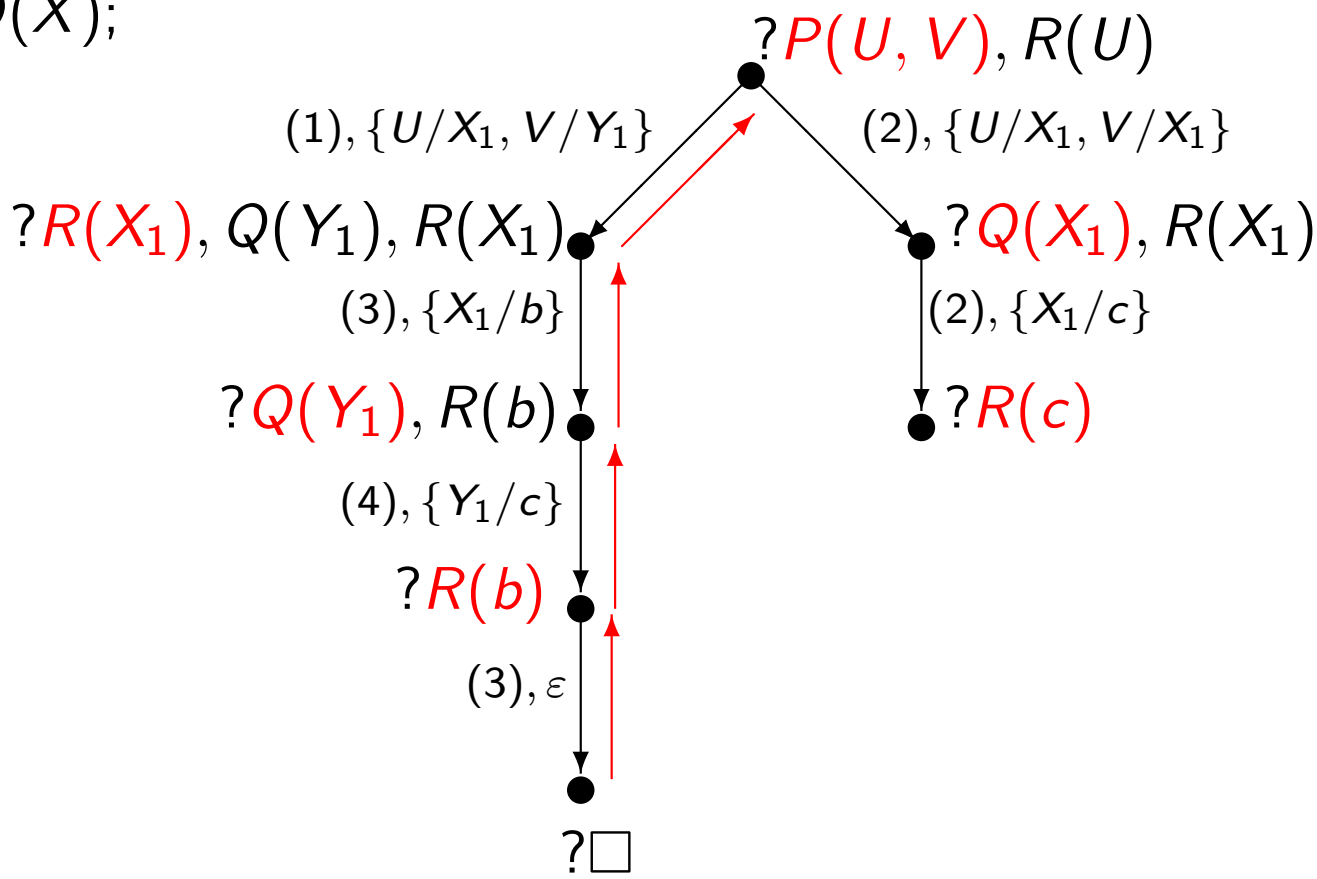
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

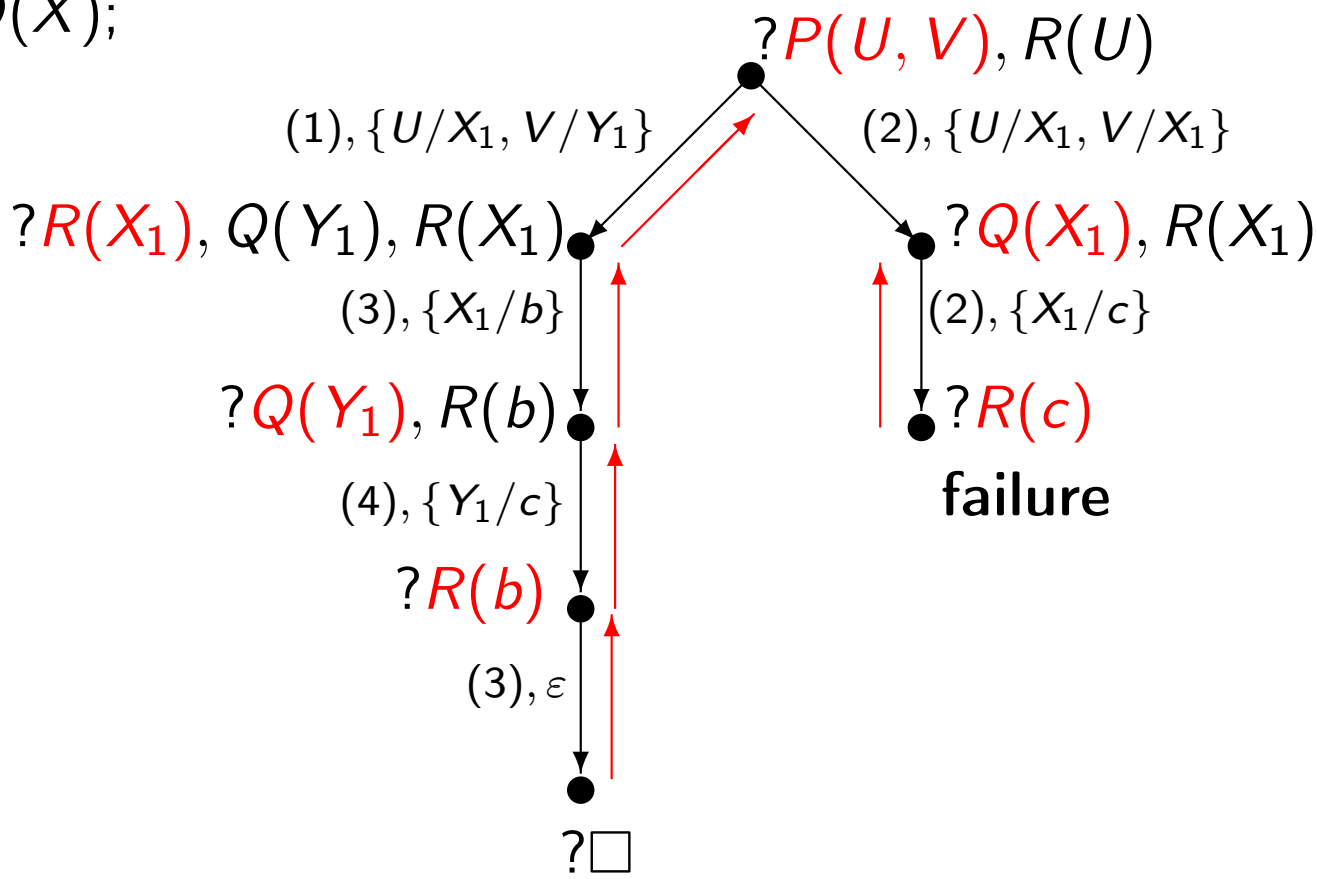
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

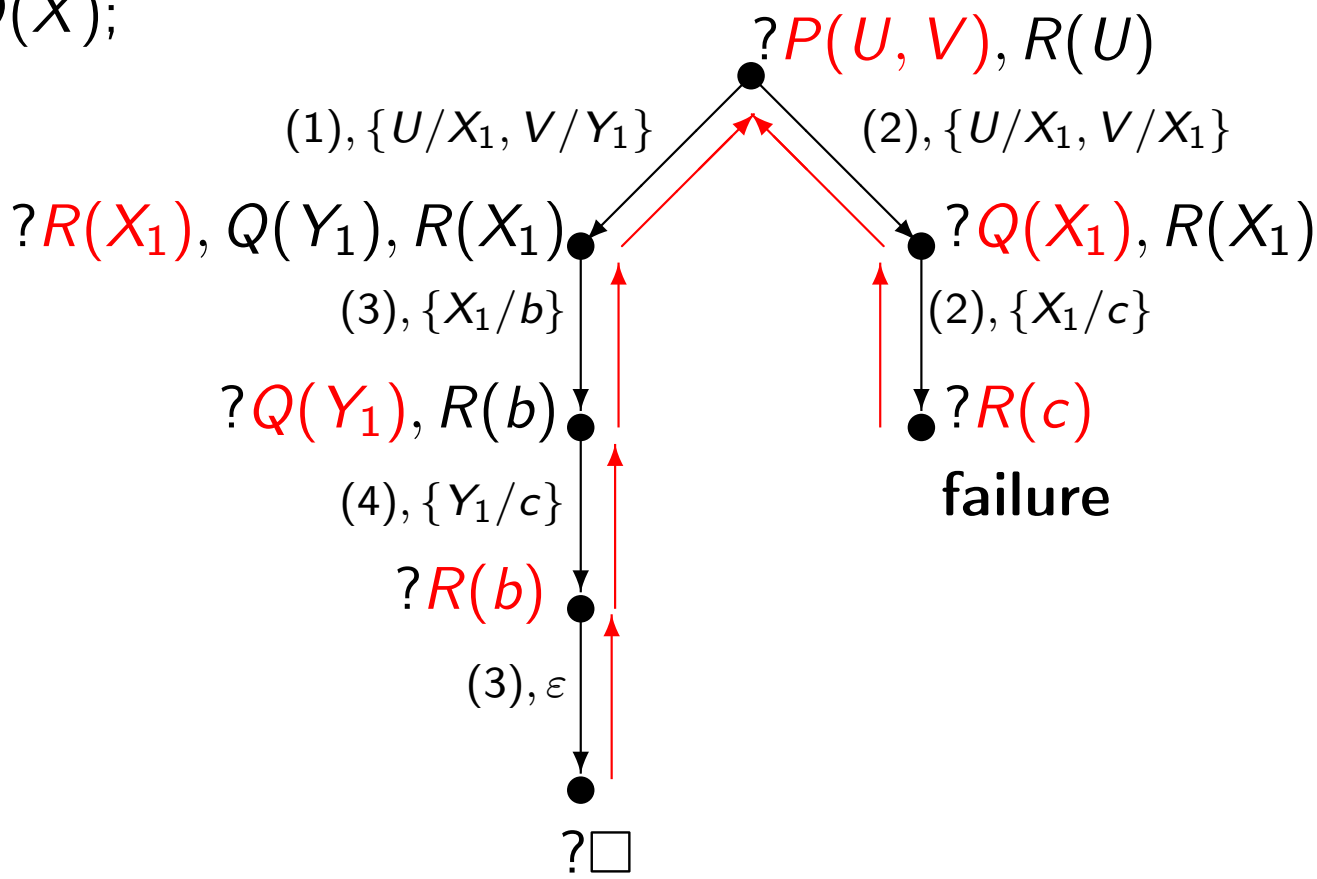
Пример обхода в глубину с возвратом.

$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$

$P(X,X) \leftarrow Q(X);$

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Стратегия обхода в глубину с возвратом

- имеет эффективную реализацию: в памяти нужно хранить лишь запросы той ветви, по которой идет обход, и каждый запрос должен вести учет использованных программных утверждений;
- является, к сожалению, вычислительно неполной.

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

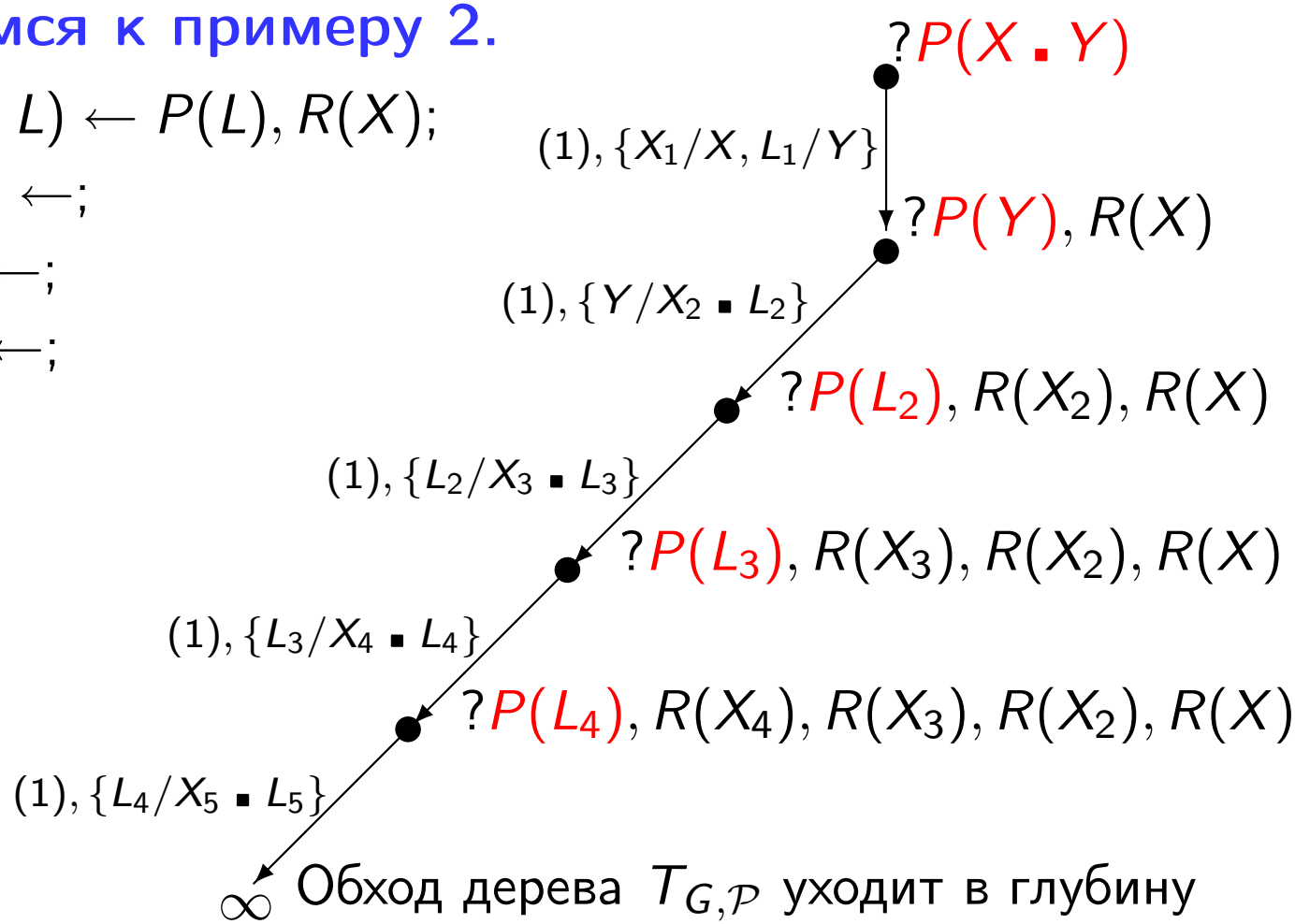
Обратимся к примеру 2.

$\mathcal{P}:\ P(X \cdot L) \leftarrow P(L), R(X);$

$P(\mathbf{nil}) \leftarrow;$

$R(a) \leftarrow;$

$R(c) \leftarrow;$



Обход дерева $T_{G,\mathcal{P}}$ уходит в глубину по бесконечной ветви и не может возвратиться, чтобы обнаружить успешное вычисление.

СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Стратегия обхода в глубину с возвратом чувствительна к порядку расположения программных утверждений в логических программах. Результат вычисления запроса может существенно измениться при перестановке программных утверждений.

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Еще раз пример 2.

$?P(X \cdot Y)$

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \cdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;

ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

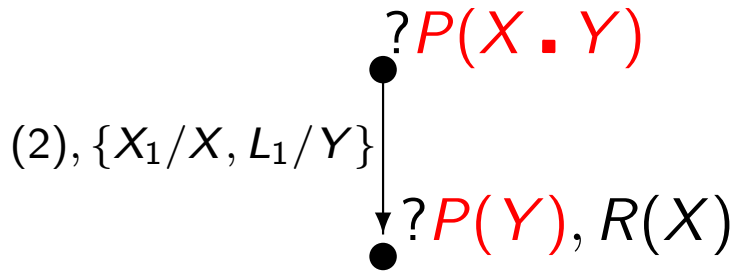
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

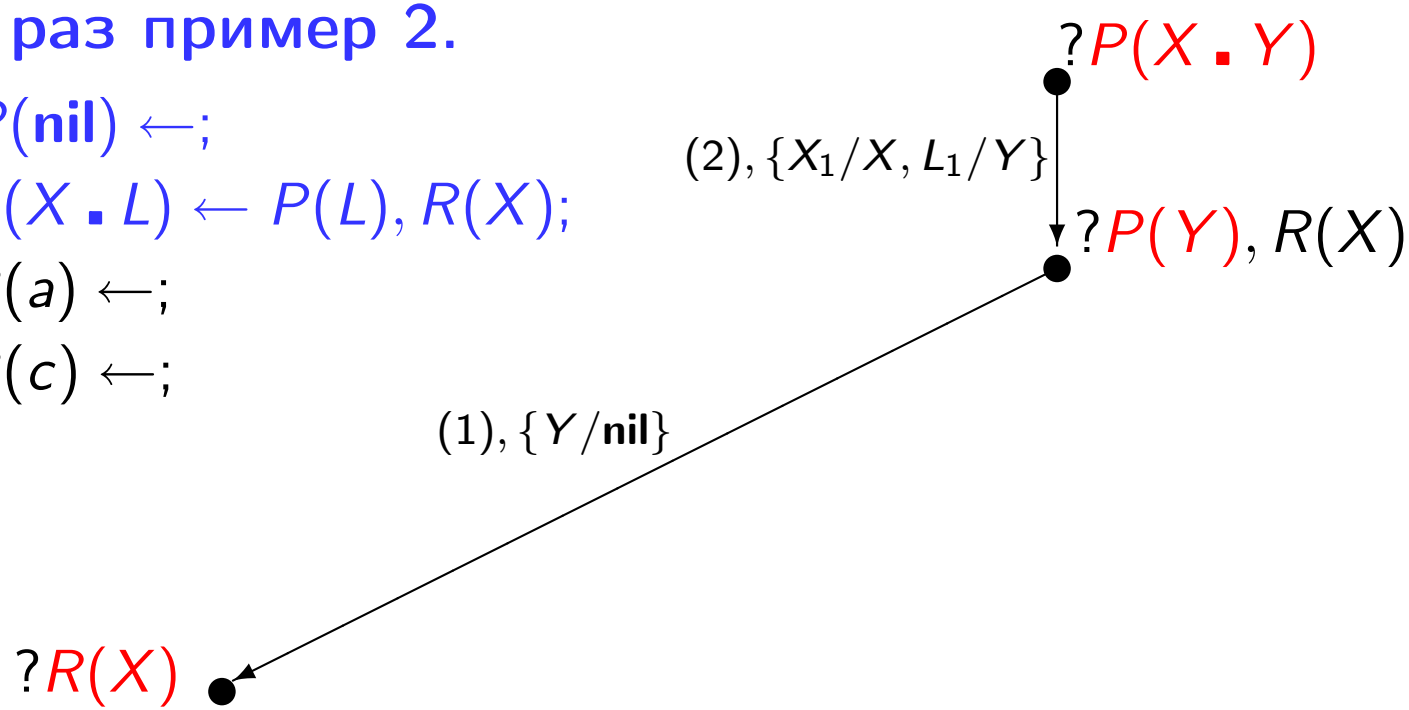
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

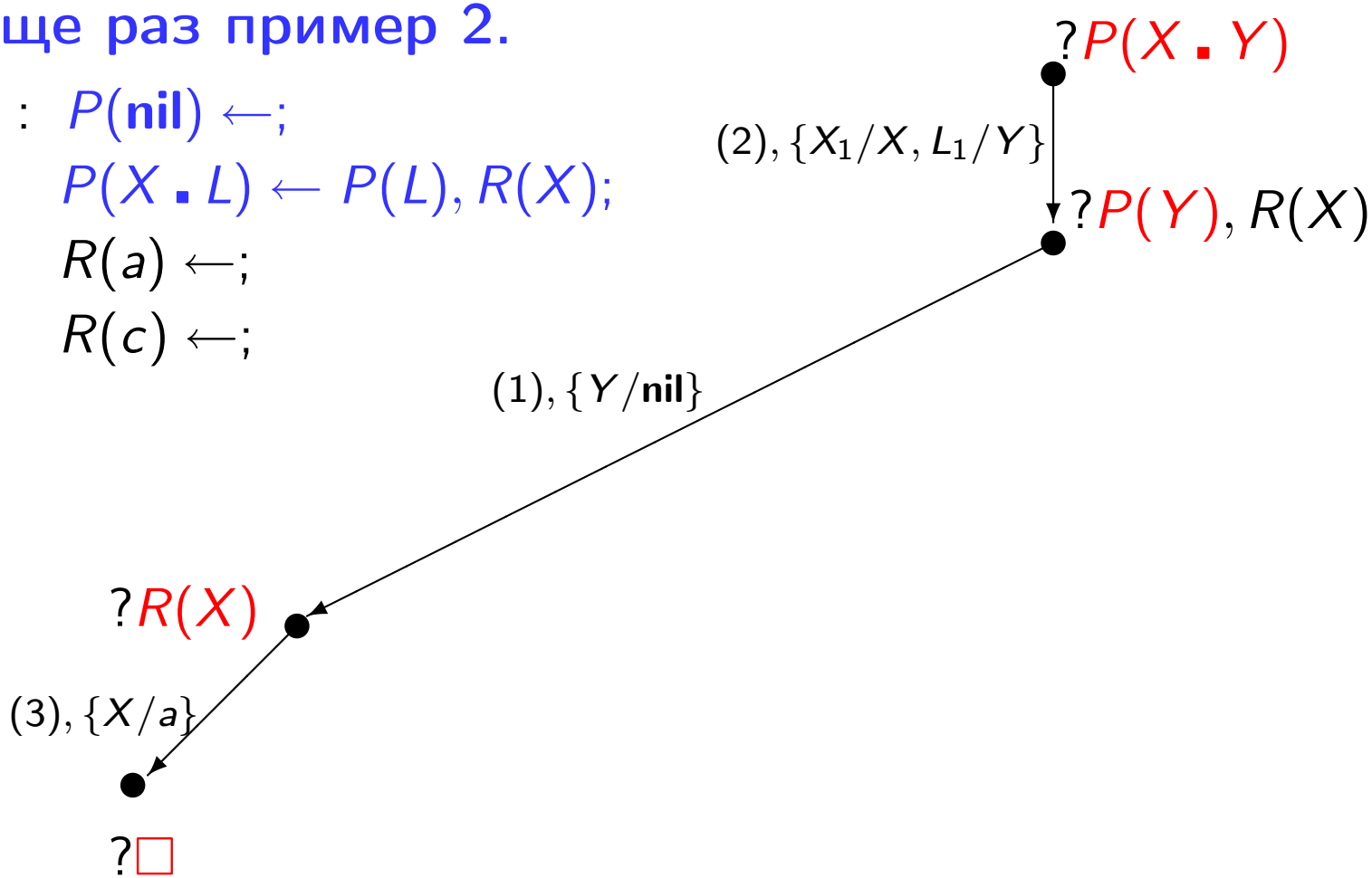
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

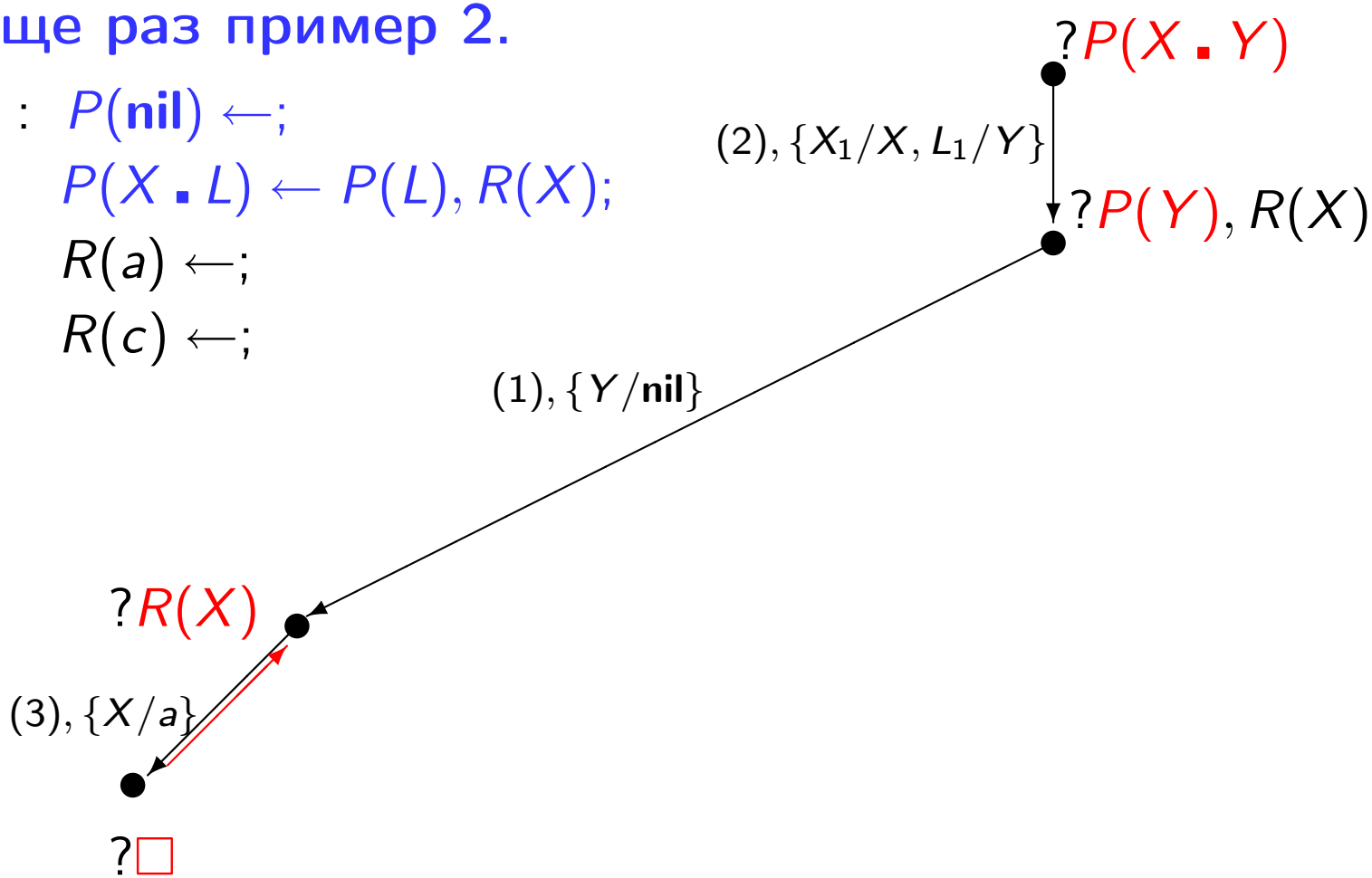
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

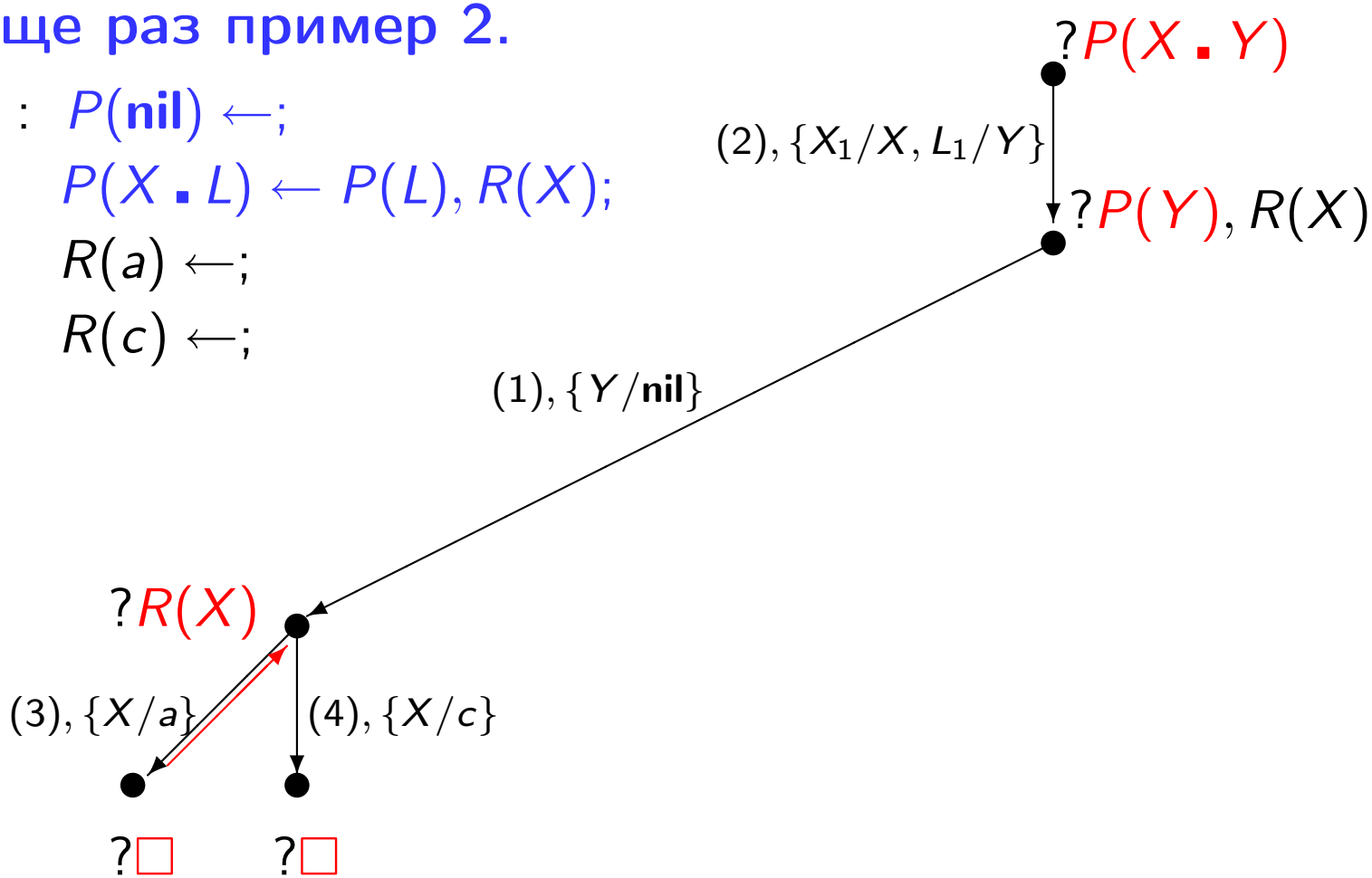
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

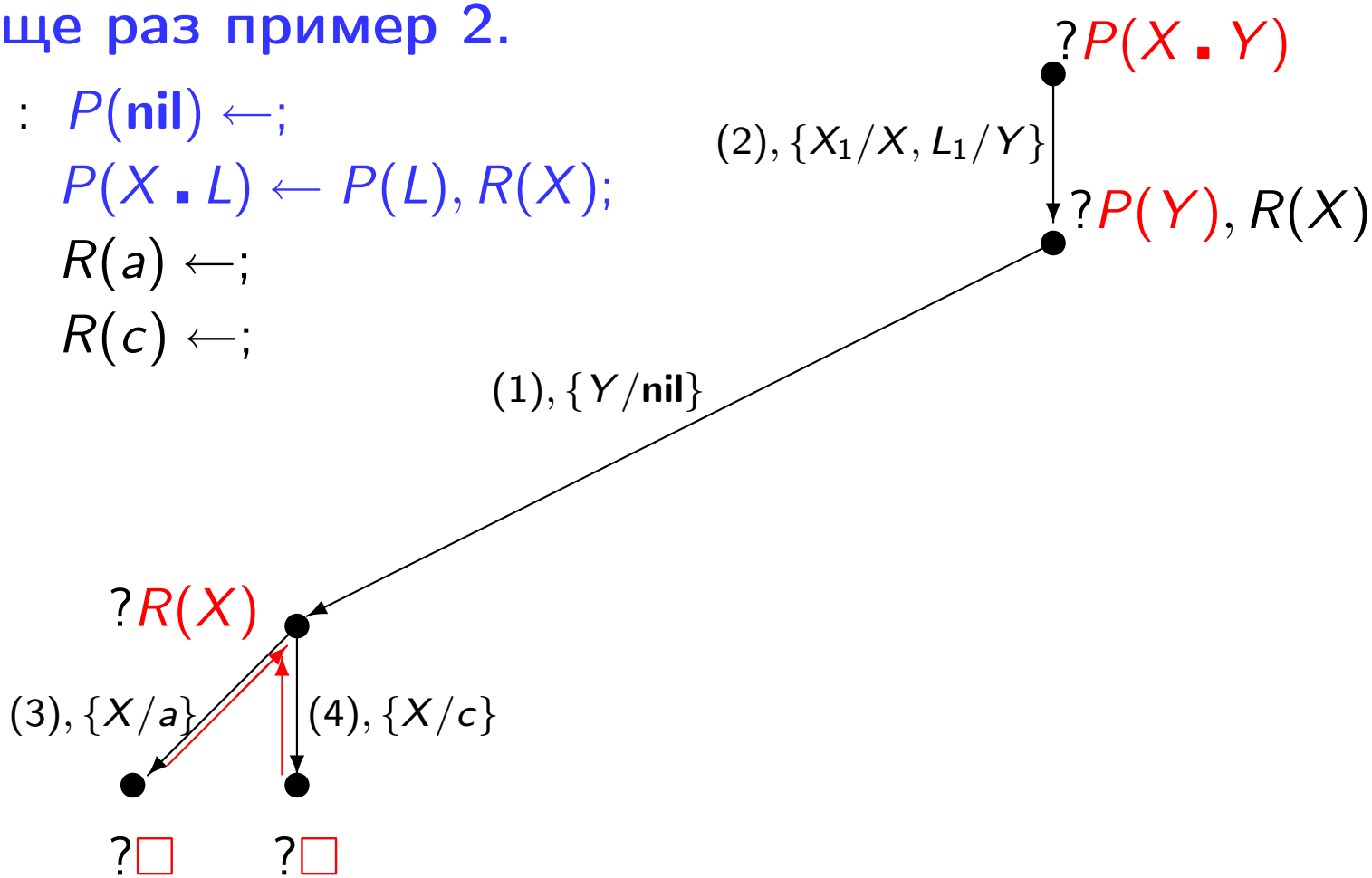
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

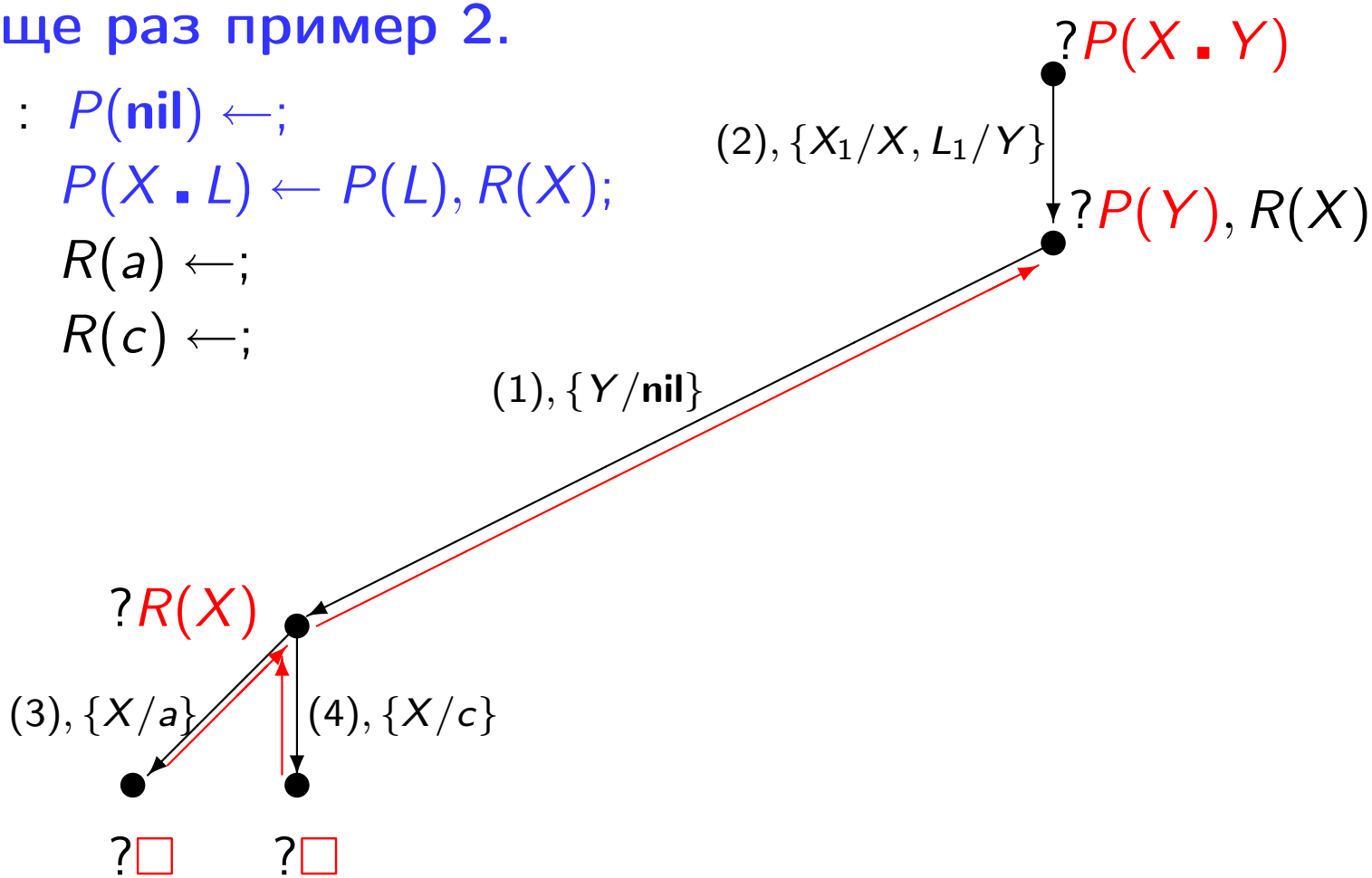
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

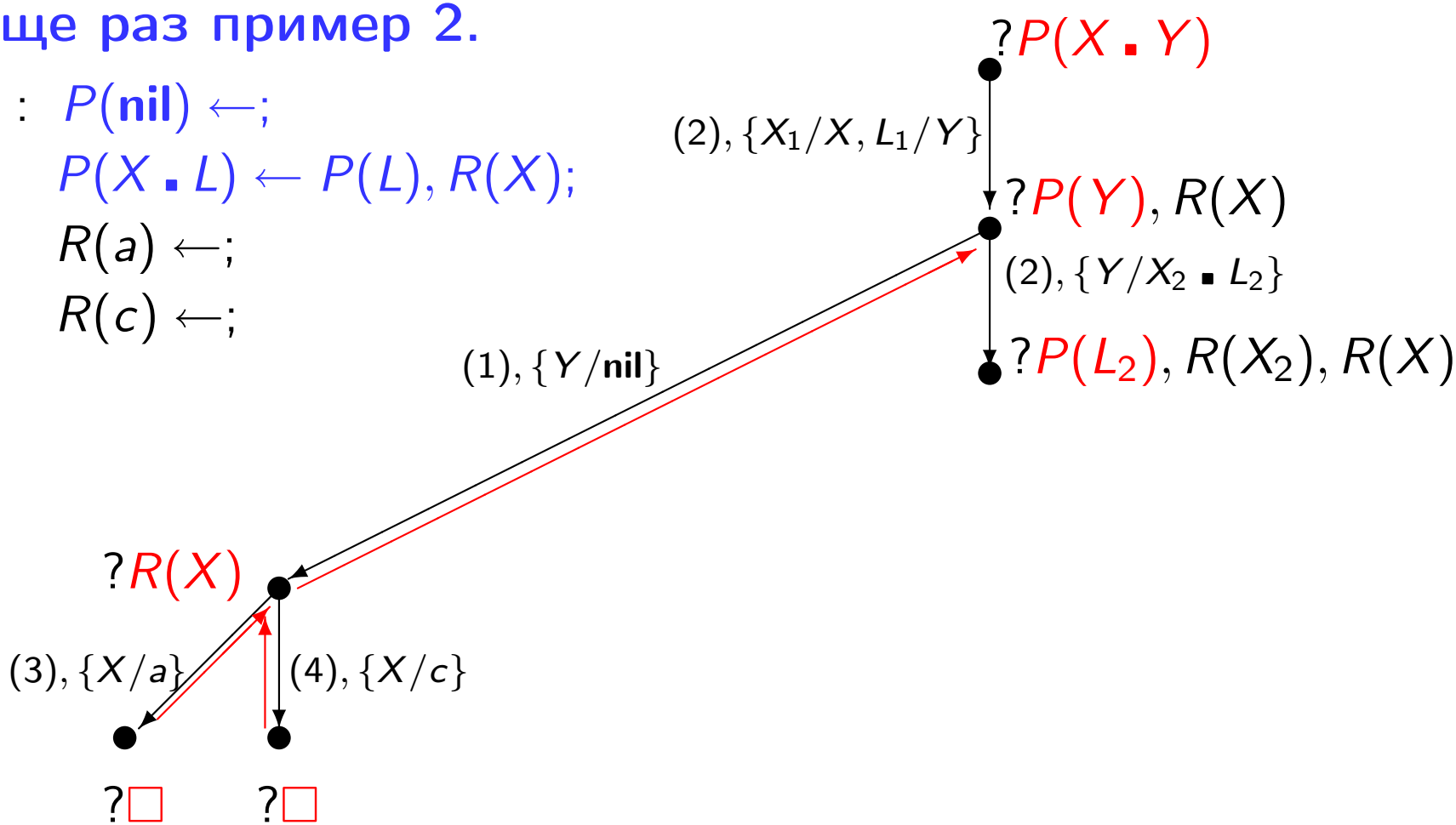
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \cdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

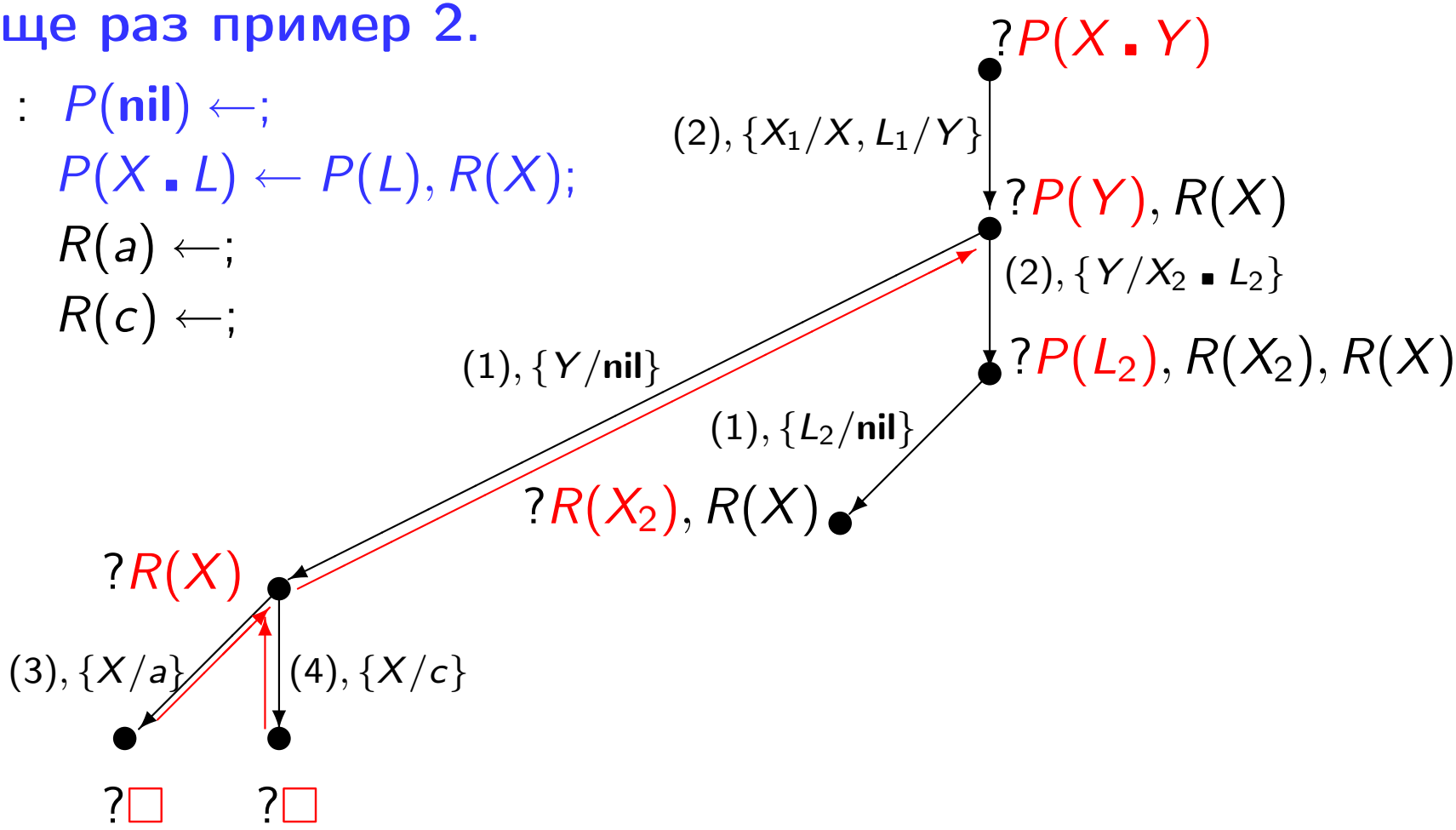
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \cdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

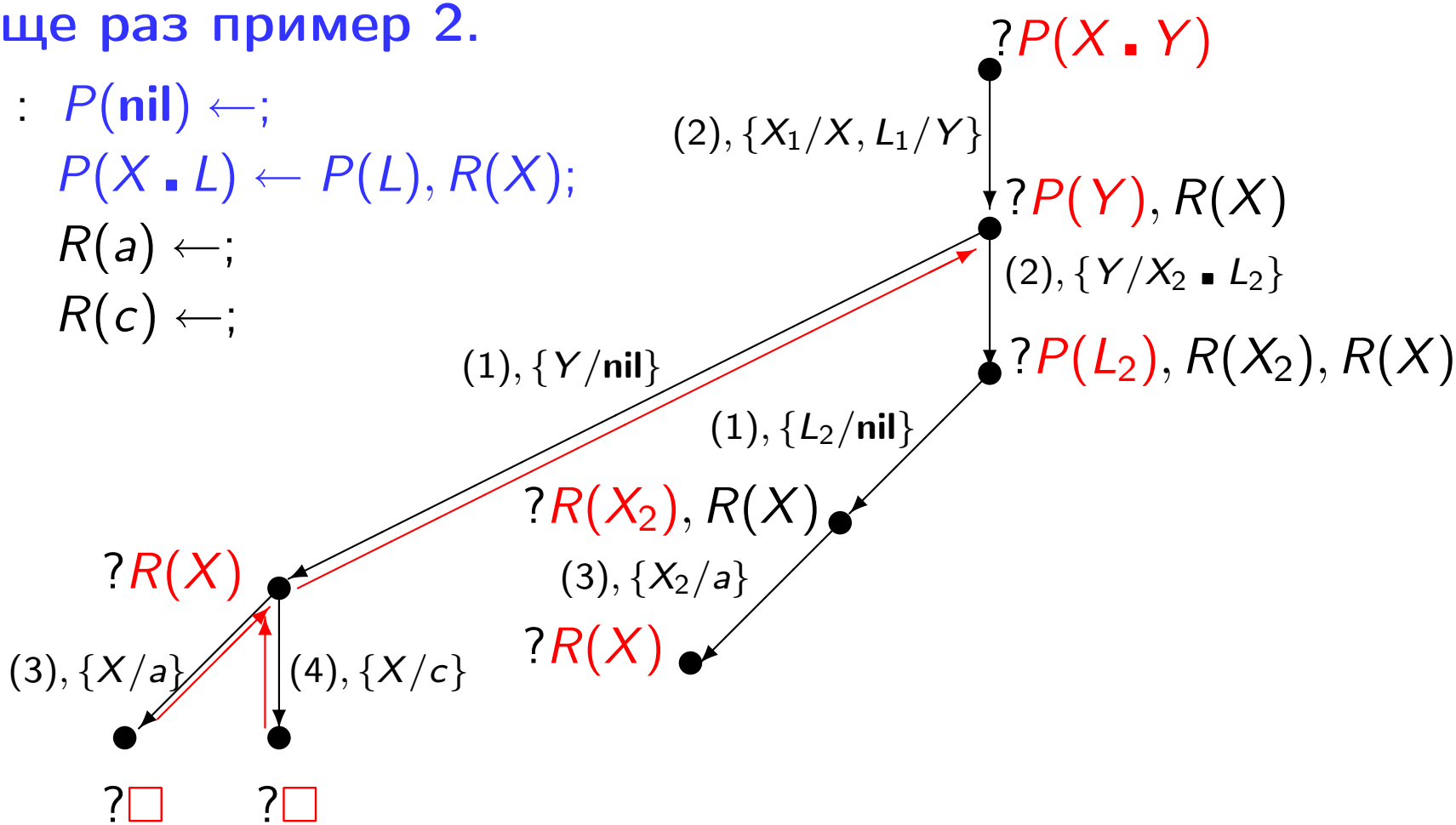
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

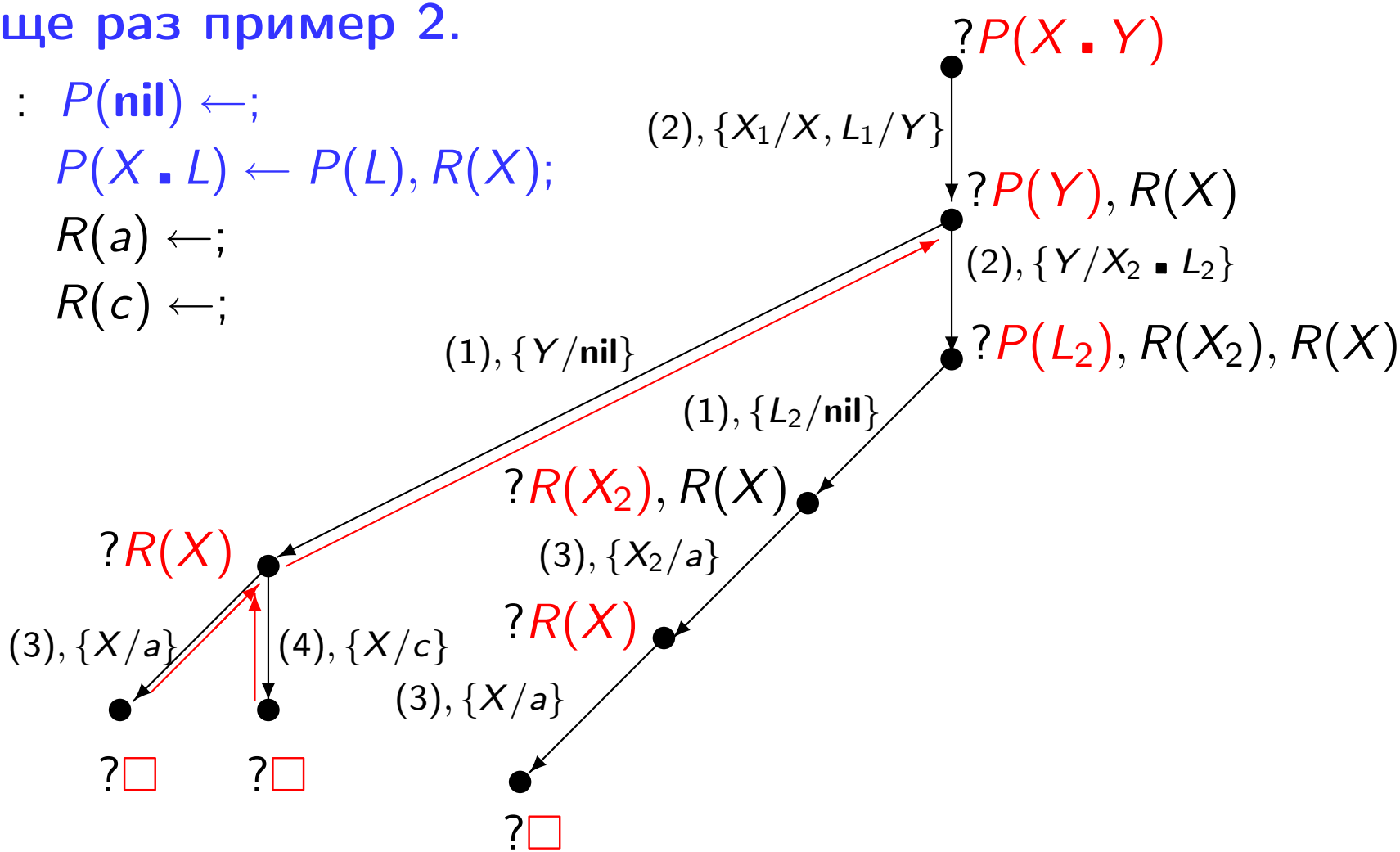
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

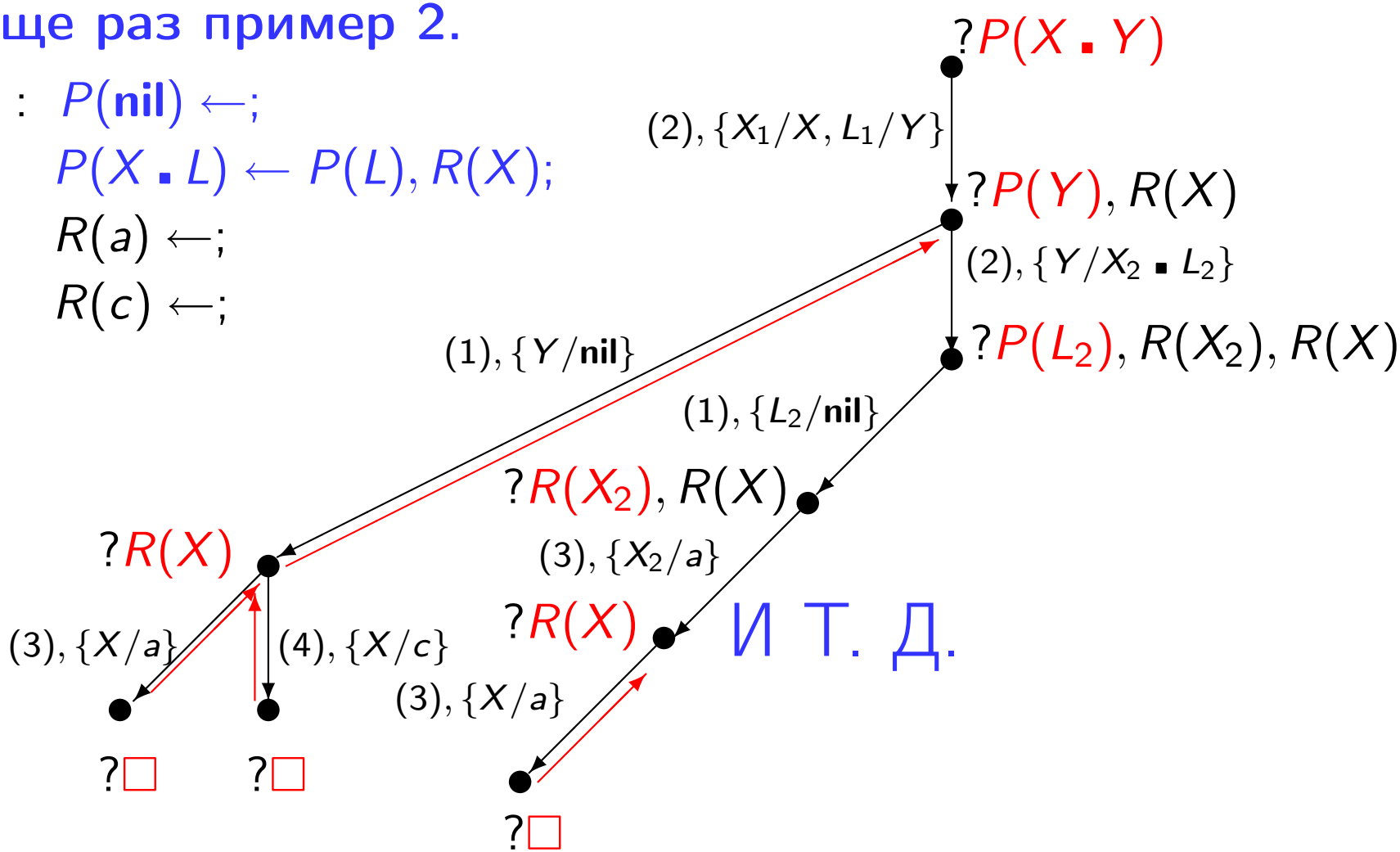
Еще раз пример 2.

$\mathcal{P}$: $P(\mathbf{nil}) \leftarrow$;

$P(X \centerdot L) \leftarrow P(L), R(X)$;

$R(a) \leftarrow$;

$R(c) \leftarrow$;



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Поскольку соображения эффективности превалируют над требованиями вычислительной полноты, в качестве **стандартной стратегии** вычисления логических программ была выбрана стратегия обхода в глубину с возвратом.

Программист должен сам позаботиться о надлежащем порядке расположения программных утверждений, чтобы стандартная стратегия вычисления позволяла отыскать все вычисленные ответы.

КОНЕЦ ЛЕКЦИИ 14.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 15.

Алгоритмическая полнота логических программ. Моделирование машин Тьюринга логическим программами. Теорема Черча.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Вопросы:

А что могут вычислять хорновские логические программы?

Какие задачи можно решать с их помощью, а какие нельзя?

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Есть много разных моделей вычислений. Одни из них проводят вычисления над строками (словами), другие — над графами, третьи — над молекулами ДНК, и т. п. Как сравнить их вычислительные способности?

Структуры данных, над которыми работают эти модели, позволяют кодировать натуральные числа. Поэтому чтобы сравнить вычислительные способности алгоритмических моделей, достаточно выяснить, какие функции натурального аргумента способны вычислять эти модели.

Исследования показали, что все известные **алгоритмические модели** способны вычислять только **частично-рекурсивные функции** натурального аргумента. Это открытие дало основание многим математикам (Тьюринг, Черч, Клини, Гедель, Марков, и др.) выдвинуть следующий тезис.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Тезис Черча

Класс эффективно (алгоритмически) вычислимых арифметических функций в точности совпадает с классом арифметических функций, вычислимых в каждой из перечисленных ниже моделей вычислений

- машины Тьюринга—Поста,
- λ-исчисление Черча—Клини,
- системы равенств Эрбрана—Геделя,
- алгорифмы Маркова,
- системы Колмогорова—Шенхаге,
- машины Минского,
- ...

Модели вычислений такого вида называются **алгоритмически полными** . Найдется ли место в этом ряду для хорновских логических программ?

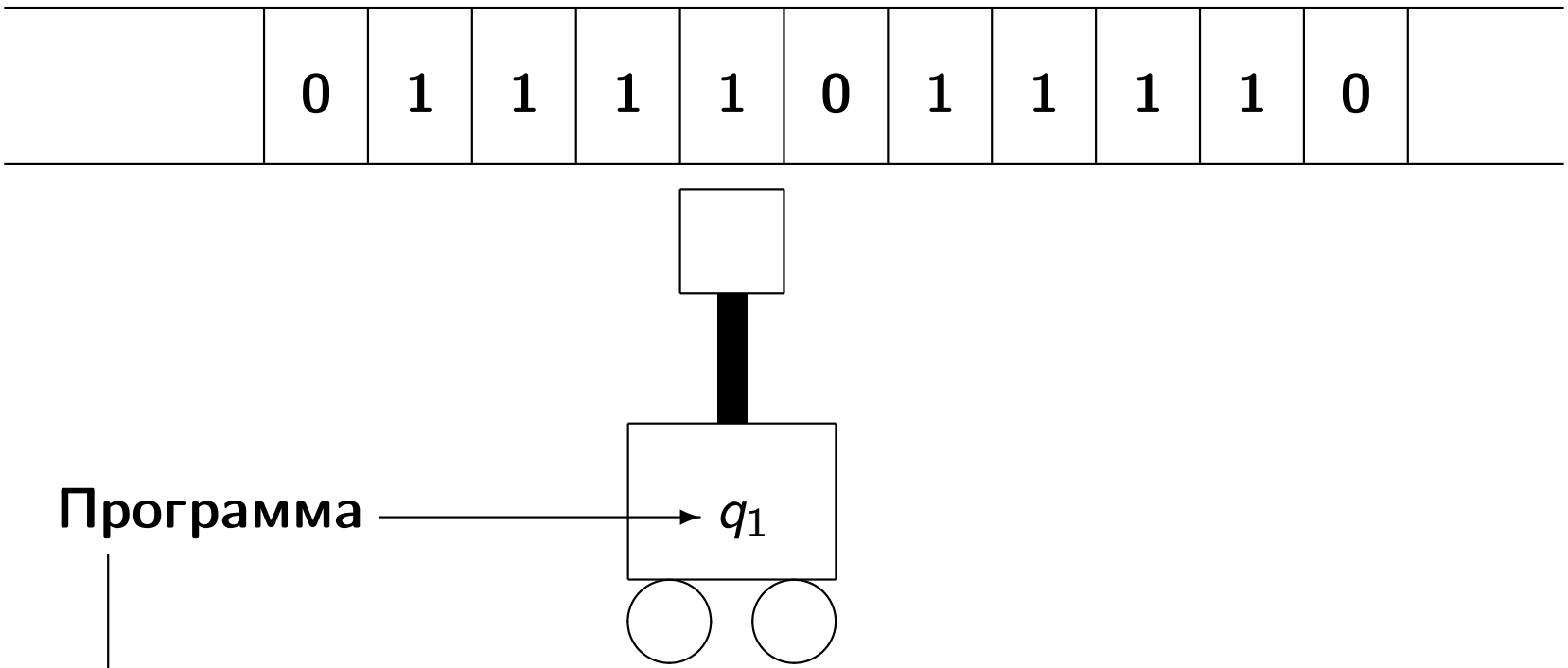
АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Для того чтобы убедиться в алгоритмической полноте хорновских логических программ, достаточно взять любую из перечисленных алгоритмически полных моделей вычислений (например, машины Тьюринга) и показать, что для любой программы П в выбранной модели найдется подходящая логическая программа $\mathcal{P}_{\text{П}}$, воспроизводящая (моделирующая) вычисления программы П.

Итак, вспомним, как устроены машины Тьюринга, и попробуем построить логическую программу-интерпретатор машин Тьюринга.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

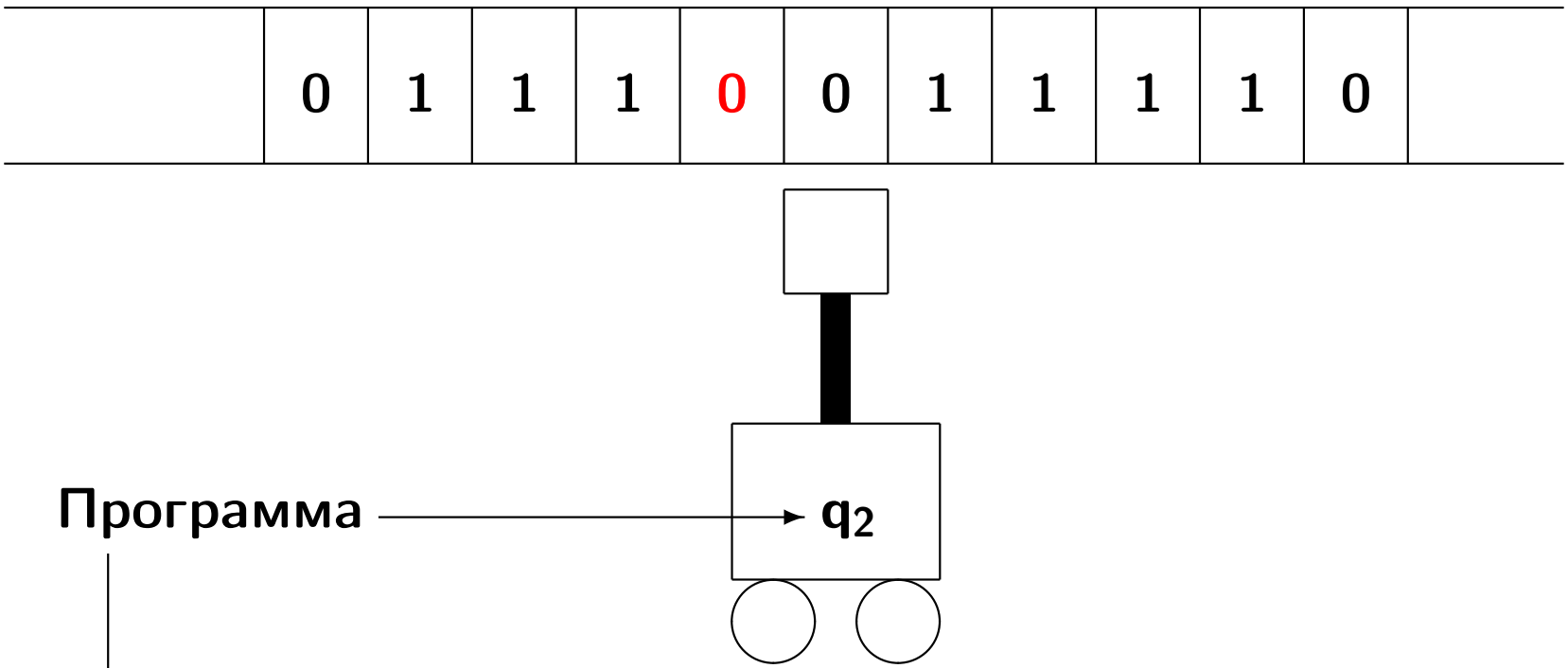


Команды:

увидев "1", стереть его, записать "0"и сдвинуться вправо

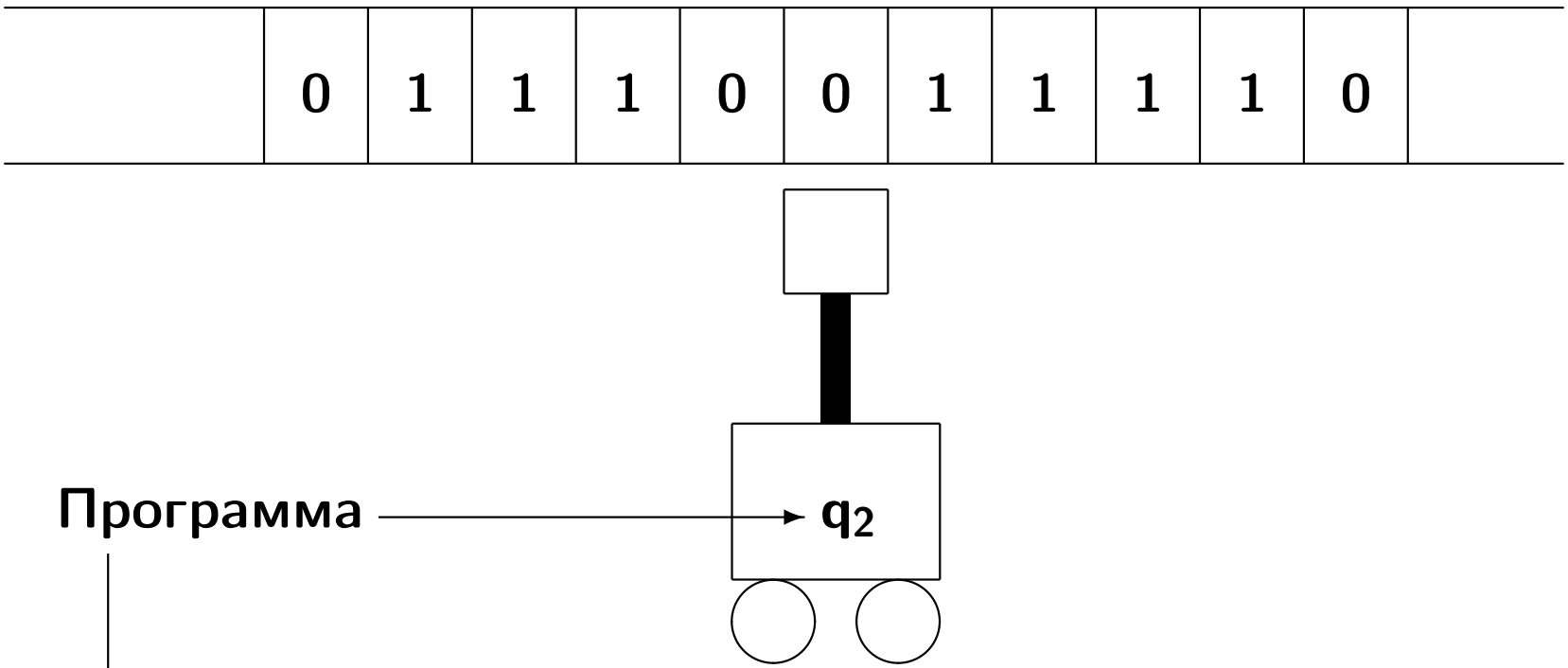
АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга



АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

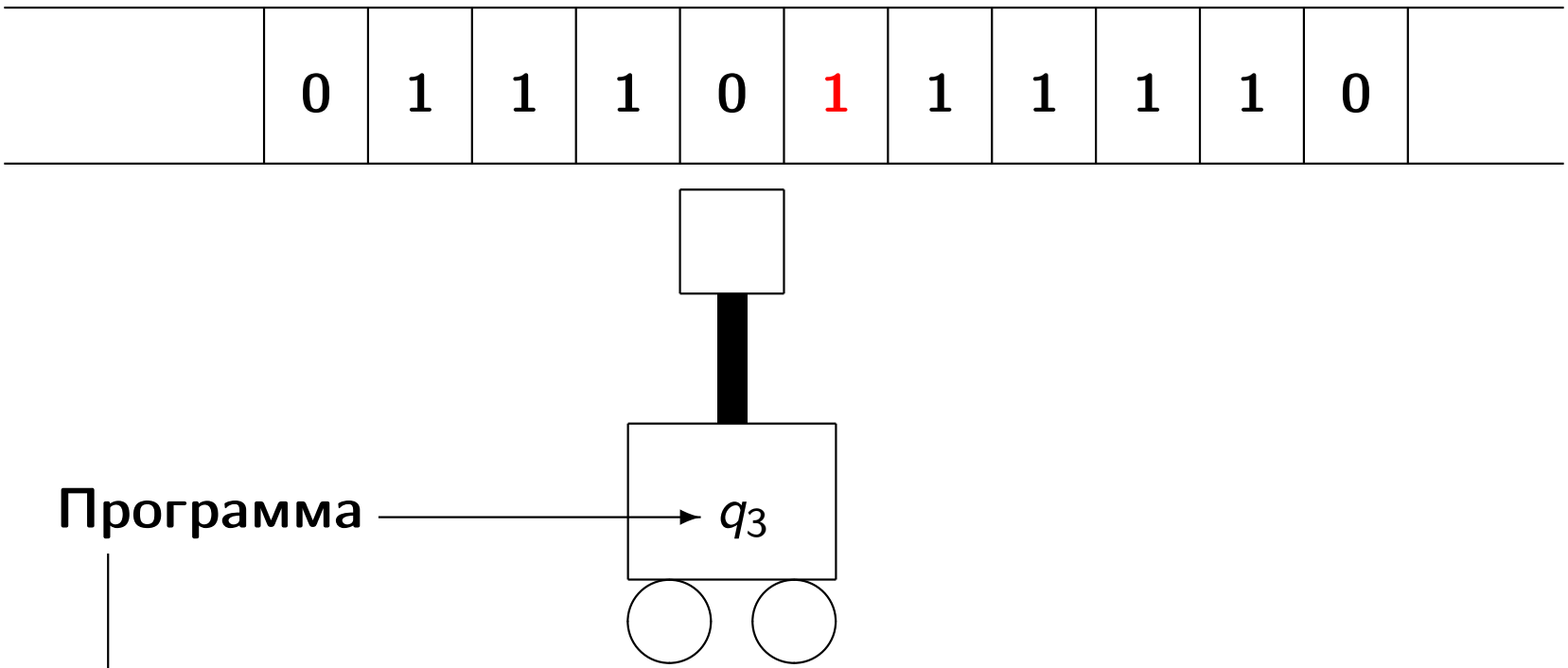


Команды:

увидев "0", стереть его, записать "1"и сдвинуться влево

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга



АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Более строго модель вычислений машин Тьюринга описывается так.

Задан ленточный алфавит $\mathcal{A} = \{a_0, a_1, a_2, \ldots, a_n\}$, в котором особо выделен одна из букв a_0 (пустой символ).

Ленточным словом называется всякое слово в алфавите $\mathcal{A}$. Множество всех ленточных слов обозначим $\mathcal{A}^*$. Для каждого слова $w = z_1 z_2 \ldots z_{n-1} z_n$ будем использовать запись w^{-1} для обозначения обратного слова $z_n z_{n-1} \ldots z_2 z_1$.

Задан алфавит состояний $\mathcal{Q} = \{q_0, q_1, q_2, \ldots, q_m\}$, в котором особо выделено одно из состояний q_0 (начальное состояние).

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Ленточной конфигурацией называется всякое слово вида

$$w'\ q\ x\ w'', \qquad \text{где}$$

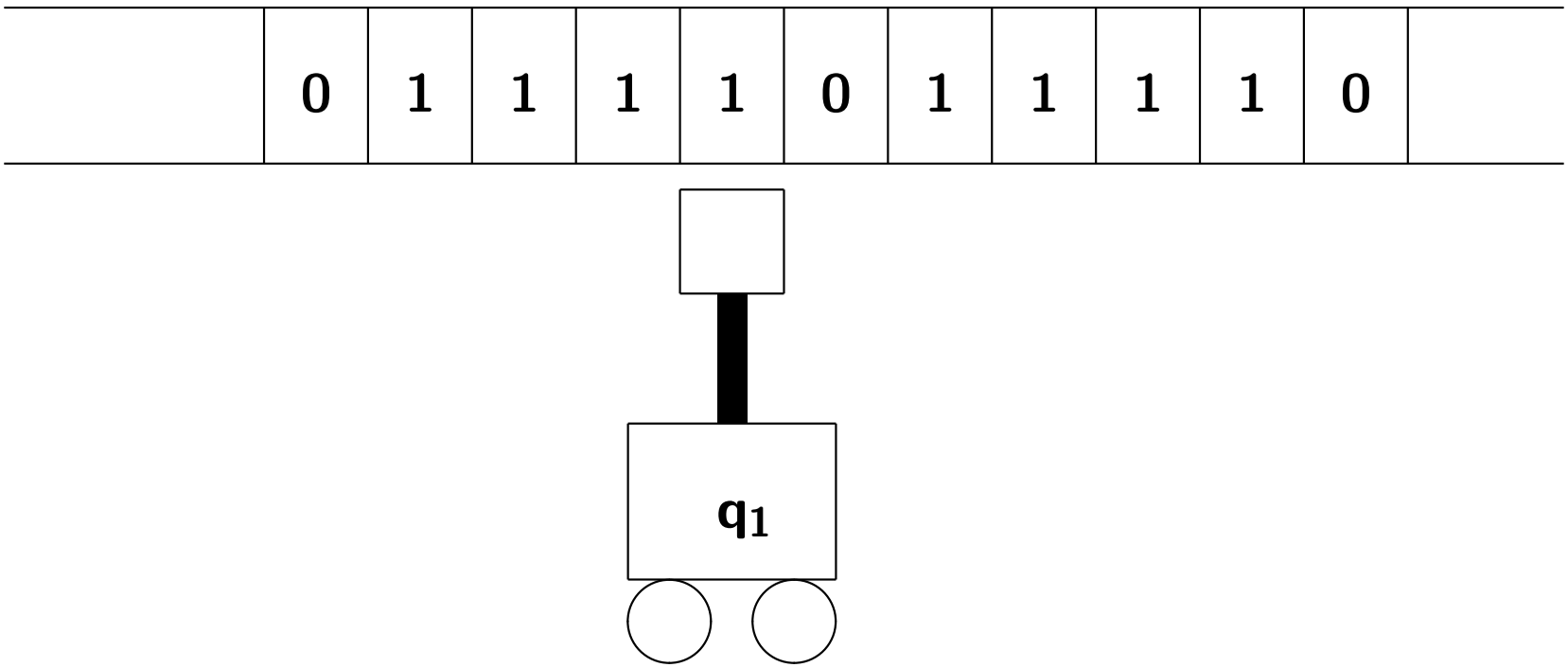
- q — состояние, $q \in \mathcal{Q}$,
- x — ленточная буква, $x \in \mathcal{A}$,
- w', w'' — ленточные слова, $w', w'' \in \mathcal{A}^*$.

- q — это то состояние, в котором находится МТ,
- x — это буква, которая записана в той ячейке ленты, которую обозревает считывающая головка МТ,
- w' — это ленточное слово, составленное из символов, записанных **слева** от обозреваемой ячейки,
- w'' — это ленточное слово, составленное из символов, записанных **справа** от обозреваемой ячейки.

По умолчанию считается, что во всех остальных ячейках ленты записаны пустые символы.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга



Ленточная конфигурация: $0111q_11011110$

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Ленточная конфигурация вида $u\ q_0\ x\ v$ называется начальной конфигурацией .

Множество всех конфигураций обозначим $Conf_{\mathcal{A},Q}$.

Множество всех начальных конфигураций обозначим $Conf^0_{\mathcal{A},Q}$.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Командой называется всякая пятерка вида

$$q\ x\ y\ q'\ D, \qquad \text{где}$$

- q, q' — состояния, $q, q' \in \mathcal{Q}$,
- x, y — ленточные буквы, $x, y \in \mathcal{A}$,
- D — направление сдвига головки, $D \in \{L, R\}$.

Эту команду нужно понимать так:
если МТ находится в состянии q и обозревает символ x, то записать в обозреваемую ячейку символ y, перейти в состояние q' и сдвинуть считывающую головку на одну ячейку в направлении D.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Каждая команда K задает отношение перехода $\to_K$ на множестве ленточных конфигураций

$$\alpha \ \to_K \ \beta$$

Оно определяется так:

- если $\alpha = uzqxv$ и $K = qxyq'L$, то $\beta = uq'zyv$,
- если $\alpha = qxv$ и $K = qxyq'L$, то $\beta = q'a_0yv$,
- если $\alpha = uqxzv$ и $K = qxyq'R$, то $\beta = uyq'zv$,
- если $\alpha = uqx$ и $K = qxyq'R$, то $\beta = uyq'a_0$.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Программа машины Тьюринга — это произвольное множество команд $\pi = \{K_1, K_2, \ldots, K_N\}$. Программа π называется детерминированной , если для любых двух команд этой программы
$K_i = q_i x y q_i' D_i$,
$K_j = q_j z t q_j' D_j$,
выполняется хотя бы одно из двух условий $q_i \neq q_j$, $x \neq z$.

Программа π задает отношение переходов на множестве ленточных конфигураций $\rightarrow_\pi = \bigcup\limits_{K \in \pi} \rightarrow_K$.

Конфигурация α называется заключительной для программы π, если не существует никакой конфигурации β, для которых выполняется $\alpha \rightarrow_\pi \beta$. Это означает, что $\alpha = u\ q\ x\ v$, и в программе π нет ни одной команды $K = qx\ldots$.

АЛГОРИТМИЧЕСКАЯ ПОЛНОТА

Машины Тьюринга

Вычислением машины Тьюринга с программой π на начальной конфигурации $\alpha_0, \alpha_0 \in Conf^0$ называется последовательность конфигураций

$$\pi(\alpha_0) = \alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_i, \alpha_{i+1}, \ldots$$

удовлетворяющая следующим условиям:

- для любого $i, i \geq 0$, верно $\alpha_i \to_\pi \alpha_{i+1}$;
- последовательность $\pi(\alpha_0)$ либо является бесконечной, либо заканчивается заключительной конфигурацией α_N.

В последнем случае α_N называется результатом вычисления . Результат бесконечного вычисления считается неопределенным. Ясно, что вычисление $\pi(\alpha_0)$ детерминированной машины Тьюринга однозначно определяется начальной конфигурацией α_0 и программой π.

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Чтобы логические программы могли моделировать вычисления машин Тьюринга, нужно выбрать подходящие логические конструкции для представления конфигураций и команд.

Ленточные слова будем представлять списками:
слово $w = z_1 z_2 \ldots z_n$ будет представлено списком
$\mathit{list}(w) = z_1 \,.\, z_2 \,.\, \ldots \,.\, z_n \,.\, \mathbf{nil}$.

Каждая ленточная конфигурация $\alpha = u\ q\ z\ v$ будет представлена парой списков $\mathit{left}(\alpha),\ \mathit{right}(\alpha)$, где
$\mathit{left}(\alpha) = \mathit{list}(u^{-1})$,
$\mathit{right}(\alpha) = q \,.\, z \,.\, \mathit{list}(v)$.

Например,
$\mathit{left}(0111q_1 1011110) = 1 \,.\, 1 \,.\, 1 \,.\, 0 \,.\, \mathbf{nil}$,
$\mathit{right}(0111q_1 1011110) = q_1 \,.\, 1 \,.\, 0 \,.\, 1 \,.\, 1 \,.\, 1 \,.\, 1 \,.\, 0 \,.\, \mathbf{nil}$.

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Каждой команде $K = q\ b\ c\ q'\ L$ сопоставим пару программных утверждений $D_1(K),\ D_2(K)$:

$$\begin{array}{lrcl} D_1(K): & P(Z \centerdot X, q \centerdot b \centerdot Y, X', Y') & \leftarrow & P(X, q' \centerdot Z \centerdot c \centerdot Y, X', Y'); \\ D_1(K): & P(\mathbf{nil}, q \centerdot b \centerdot Y, X', Y') & \leftarrow & P(\mathbf{nil}, q' \centerdot a_0 \centerdot c \centerdot Y, X', Y'); \end{array}$$

Каждой команде $K = q\ b\ c\ q'\ R$ сопоставим пару программных утверждений $D_1(K),\ D_2(K)$:

$$\begin{array}{lrcl} D_1(K): & P(X, q \centerdot b \centerdot Z \centerdot Y, X', Y') & \leftarrow & P(c \centerdot X, q' \centerdot Z \centerdot Y, X', Y'); \\ D_1(K): & P(X, q \centerdot b \centerdot \mathbf{nil}, X', Y') & \leftarrow & P(c \centerdot X, q' \centerdot a_0 \centerdot \mathbf{nil}, X', Y'); \end{array}$$

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Кроме того, для каждой пары q, z, где $q \in \mathcal{Q}$, $z \in \mathcal{A}$, введем факт $D_{q,z}$:

$$D_{q,z} : P(X, q \blacksquare z \blacksquare Y, X, q \blacksquare z \blacksquare Y) \ \leftarrow;$$

Теперь для каждой машины Тьюринга $\pi = \{K_1, K_2, \ldots, K_N\}$ выделим множество T_π всех пар qx, которые **не являются** началами ни одной из команд программы π...

и построим хорновскую логическую программу

$$\mathcal{P}_\pi = \{D_1(K), D_2(K) \ : \ K \in \pi\} \cup \bigcup_{(qx \in T_\pi)} D_{qx}.$$

Можно надеяться, что логическая программа $\mathcal{P}_\pi$ воспроизводит все вычисления машины Тьюринга π.

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Пример

Пусть $\mathcal{A} = \{0, 1\}$ и пусть $\pi = \{K_1, K_2, K_3\}$, где
$K_1 = \ q_0\ 0\ 1\ q_1\ R,$
$K_2 = \ q_1\ 0\ 1\ q_0\ L,$
$K_3 = \ q_1\ 1\ 1\ q_1\ R.$

Машина Тьюринга с программой π транслируется в логическую программу $\mathcal{P}_\pi$.

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Пример

Логическая программа $\mathcal{P}_\pi$:

$$
\begin{array}{lcl}
P(X, q_0 \centerdot 0 \centerdot Z \centerdot Y, X', Y') & \leftarrow & P(1 \centerdot X, q_1 \centerdot Z \centerdot Y, X', Y'); \\
P(X, q_0 \centerdot 0 \centerdot \mathbf{nil}, X', Y') & \leftarrow & P(1 \centerdot X, q_1 \centerdot 0 \centerdot \mathbf{nil}, X', Y'); \\
 & & \\
P(Z \centerdot X, q_1 \centerdot 0 \centerdot Y, X', Y') & \leftarrow & P(X, q_0 \centerdot Z \centerdot 1 \centerdot Y, X', Y'); \\
P(\mathbf{nil}, q_1 \centerdot 0 \centerdot Y, X', Y') & \leftarrow & P(\mathbf{nil}, q_0 \centerdot 0 \centerdot 1 \centerdot Y, X', Y'); \\
 & & \\
P(X, q_1 \centerdot 1 \centerdot Z \centerdot Y, X', Y') & \leftarrow & P(1 \centerdot X, q_1 \centerdot Z \centerdot Y, X', Y'); \\
P(X, q_1 \centerdot 1 \centerdot \mathbf{nil}, X', Y') & \leftarrow & P(1 \centerdot X, q_1 \centerdot 0 \centerdot \mathbf{nil}, X', Y'); \\
 & & \\
P(X, q_0 \centerdot 1 \centerdot Z, X, q_0 \centerdot 1 \centerdot Z) & \leftarrow & ;
\end{array}
$$

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Лемма 1

Для любой пары ленточных конфигураций $\alpha, \beta \in Conf$ и команды K отношение перехода $\alpha \rightarrow_K \beta$ выполняется тогда и только тогда, когда запрос
$G_\alpha : ?P(left(\alpha), right(\alpha), X, Y)$
и одно из программных утверждений $D_1(K),\ D_2(K)$
имеют SLD-резольвенту $G_\beta :?P(left(\beta), right(\beta), X, Y)$.

Доказательство

Самостоятельно.

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Лемма 2

Ленточная конфигурация $\alpha \in Conf$ является заключительной для машины Тьюринга π тогда и только тогда, когда запрос $G_\alpha : ?P(left(\alpha), right(\alpha), X, Y)$
и одно из программных утверждений множества $\bigcup\limits_{(qx \in T_\pi)} D_{qx}$
имеют пустой дизъюнкт $\square$ в качестве SLD-резольвенты.

Доказательство

Самостоятельно.

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Теорема (о моделировании МТ логическими программами)

Каковы бы ни были машина Тьюринга π и начальная конфигурация α_0, вычисление

$\alpha_0 \to_\pi \alpha_1 \to_\pi \alpha_2 \to_\pi \cdots \to_\pi \alpha_N$

завершается заключительной конфигурацией α_N тогда и только тогда, когда запрос

$G_{\alpha_0} : ?R(\mathit{left}(\alpha_0), \mathit{right}(\alpha_0), X, Y)$

к хорновской логической программе $\mathcal{P}_\pi$ имеет успешное вычисление с вычисленным ответом

$\theta = \{X/\mathit{left}(\alpha_N),\ Y/\mathit{right}(\alpha_N)\}.$

Доказательство

Следует из лемм 1 и 2. □

МОДЕЛИРОВАНИЕ МАШИН ТЬЮРИНГА ЛОГИЧЕСКИМИ ПРОГРАММАМИ

Таким образом, хорновские логические программы обладают не меньшими вычислительными воможностями, чем машины Тьюринга. Значит, логическое программирование — это универсальная модель вычислений, позволяющая вычислять все эффективно вычислимые функции.

Но это также означает, что логическому программированию присущи все те трудности анализа поведения программ, которые присущи и другим универсальным моделям вычислений. Речь идет об алгоритмически неразрешмых задачах.

Например, интересен вопрос о том, можно ли по заданному запросу G к логической программе $\mathcal{P}$ выяснить, имеет ли этот запрос хотя бы одно успешное вычисление. Тогда не пришлось бесполезно тратить время на построение дерева SLD-резолютивных вычислений.

ТЕОРЕМА ЧЕРЧА

Теорема Тьюринга об алгоритмической неразрешимости проблемы останова

Проблема останова машин Тьюринга алгоритмически неразрешима, т. е. не существует алгоритма (машины Тьюринга), способного вычислить следующую функцию

$$F(x,y) = \begin{cases} 1, & \text{если } x \text{ — это начальная ленточная конфигурация,} \\ & \text{если } y \text{ — это список команд МТ } \pi, \\ & \text{и вычисление } \pi(x) \text{ конечно;} \\ 0 & \text{в противном случае.} \end{cases}$$

Доказательство

Известно каждому первокурснику. □

ТЕОРЕМА ЧЕРЧА

Из теоремы о моделировании машин Тьюринга логическими программами и теоремы об алгоритмической неразрешимости проблемы останова машин Тьюринга получаем несколько важных следствий

Следствие 1.

Не существует алгоритма, способного определить по заданному запросу G к хорновской логической программе $\mathcal{P}$,

- является ли дерево SLD-резолютивных вычислений запроса G конечным;
- содержит ли дерево SLD-резолютивных вычислений запроса G хотя бы одно успешное вычисление;
- является ли заданная подстановка θ вычисленным ответом на запрос G.

ТЕОРЕМА ЧЕРЧА

Следствие 2 (Теорема Черча).

Не существует алгоритма, способного определить по заданной замкнутой формуле логики предикатов φ, является ли эта формула общезначимой, т. е. проблема общезначимости "$\models \varphi$?"алгоритмически неразрешима.

Доказательство

Пусть $G : ?C_1, \ldots, C_m$ произвольный запрос к произвольной логической программе $\mathcal{P} = \{D_1, \ldots, D_N\}$.
Тогда...

ТЕОРЕМА ЧЕРЧА

Доказательство

Запрос G к программе $\mathcal{P}$ имеет хотя бы одно успешное SLD-резолютивное вычисление

$$\Longleftrightarrow \qquad \text{(теоремы корректности и полноты)}$$

Запрос G к программе $\mathcal{P}$ имеет хоть один правильный ответ θ

$$\Longleftrightarrow \qquad \text{(определение правильного ответа)}$$

$$\{D_1, \ldots, D_N\} \models \forall z_1 \ldots \forall z_k (C_1 \& \ldots \& C_m)\theta$$

$$\Longleftrightarrow \qquad \text{(теорема о логическом следовании)}$$

$$\models D_1 \& \ldots \& D_N\} \forall z_1 \ldots \forall z_k (C_1 \& \ldots \& C_m)\theta.$$

$\square$

ТЕОРЕМА ЧЕРЧА

Теорема Черча об алгоритмической неразрешимости проблемы общезначимости показывает, что ни одна система автоматического доказательства теорем не может гарантировать решение следующих вопросов для произвольных формул:

- является ли заданная формула φ общезначимой?
- является ли заданная формула φ выполнимой?
- является ли заданная формула φ логическим следствием заданного множества формул Г?

КОНЕЦ ЛЕКЦИИ 15.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 16.

Управление вычислениями логических программ. Оператор отсечения.

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Вопросы:

А как организовано вычисление логических программ на компьютере?

Как устроен интерпретатор логических программ?

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Простейший способ организации работы логических программ на основе стандартной стратегии обхода дерева SLD-резолютивных вычислений в глубину с возвратом — это работа со **стеком** (или **магазином**).
В каждом элементе S_n стека содержится следующая информация:

- Текущее целевое утверждение (запрос) G_n;
- Композиция всех ранее вычисленных унификаторов $\eta_n = (\theta_1 \dots \theta_n)|_{\text{целев.перем}}$;
- Счетчик использованных программных утверждений $count_n$;
- Специальные пометки (о некоторых из них будет рассказано далее).

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$
\begin{array}{ll}
P(X, Y) \leftarrow R(X), Q(Y); & (1) \\
P(X, X) \leftarrow Q(X); & (2) \\
R(b) \leftarrow; & (3) \\
Q(c) \leftarrow; & (4)
\end{array}
$$

Протокол:

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$

$\eta = \varepsilon; \quad count = 1$

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$
$\eta = \varepsilon; \quad count = 1$

Протокол:

SLD-резолюция

$HOY = \{X_1/U, Y_1/V\}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$
$\eta = \varepsilon; \quad count = 1$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$
$\eta = \varepsilon; \quad count = 2$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$
$\eta = \varepsilon; \quad count = 3$

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 1$

Протокол:

SLD-резолюция

$HOY = \{U/b\}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 1$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 2$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 3$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

SLD-резолюция

$HOУ = \{V/c\}$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 1$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

Нет унификатора

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 1$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

Нет унификатора

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 2$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

Унификация

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

SLD-резолюция

$HOУ = \varepsilon$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$
$\square$ $\eta = \{U/b, V/c\}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

Успех: $\eta = \{U/b, V/c\}$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$
$\square$ $\eta = \{U/b, V/c\}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

Откат

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

Протокол:

Нет унификатора

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$
$?R(b)$ $\eta = \{U/b, V/c\}; count = 4$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 1$
$?R(U), Q(V), R(U)$
$\eta = \varepsilon; \quad count = 3$

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon;$ $\quad count = 1$
$?R(U), Q(V), R(U)$
$\eta = \varepsilon;$ $\quad count = 4$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$

$\eta = \varepsilon; \quad count = 1$

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$

$\eta = \varepsilon; \quad count = 2$

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$
$\eta = \{U/X_1, V/X_1\}; count = 1$

Протокол:

SLD-резолюция

$HOY = \{U/X_1, V/X_1\}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$
$\eta = \{U/X_1, V/X_1\}; count = 1$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$
$\eta = \{U/X_1, V/X_1\}; count = 2$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$
$\eta = \{U/X_1, V/X_1\}; count = 3$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$
$\eta = \{U/X_1, V/X_1\}; count = 4$

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$ $\eta = \{U/X_1, V/X_1\}; count = 4$
$?R(c)$ $\eta = \{U/c, V/c\}; count = 1$

Протокол:

SLD-резолюция

$HOУ = \{X_1/c\}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$ $\eta = \{U/X_1, V/X_1\}; count = 4$
$?R(c)$ $\eta = \{U/c, V/c\}; count = 1$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$ $\eta = \{U/X_1, V/X_1\}; count = 4$
$?R(c)$ $\eta = \{U/c, V/c\}; count = 2$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$ $\eta = \{U/X_1, V/X_1\}; count = 4$
$?R(c)$ $\eta = \{U/c, V/c\}; count = 3$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$ $\eta = \{U/X_1, V/X_1\}; count = 4$
$?R(c)$ $\eta = \{U/c, V/c\}; count = 4$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$
$\eta = \varepsilon; \quad count = 2$
$?Q(X_1), R(X_1)$
$\eta = \{U/X_1, V/X_1\}; count = 4$

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$

$\eta = \varepsilon; \quad count = 2$

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$

$\eta = \varepsilon; \quad count = 3$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$

$?P(U, V), R(U)$

$\eta = \varepsilon; \quad count = 4$

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример стекового вычисления логических программ

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$
\begin{array}{ll}
P(X, Y) \leftarrow R(X), Q(Y); & (1) \\
P(X, X) \leftarrow Q(X); & (2) \\
R(b) \leftarrow; & (3) \\
Q(c) \leftarrow; & (4)
\end{array}
$$

Протокол:

Конец вычислений

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Д. Уоррен предложил более эффективную схему реализации интерпретаторов логических программ (Warren Abstract Machine), в которой вместо единого стека используется несколько стеков, перераспределяющих данные вычисления наиболее оптимальным способом. Эта схема положена в основу всех современных компиляторов логических программ.

А как программист может управлять вычислением логической программы?

Есть два основных способа управления:

- Выбирать правильный порядок расположения атомов в телах процедур (по принципу: вначале решать простые задачи, а потом сложные).
- Выбирать правильный порядок расположения программных утверждений (по принципу: вначале предлагать простые способы решения, а потом сложные).

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример

Рассмотрим две программы поиска маршрута в ориентированном графе.

$$\mathcal{P}_1: \quad Path((X \,.\, Z \,.\, \mathbf{nil}) \,.\, U, X, Y) \leftarrow Path(U, Z, Y), Arc(X \,.\, Z \,.\, \mathbf{nil});$$
$$Path(nil, X, X) \leftarrow;$$

$$\mathcal{P}_2: \quad Path(nil, X, X) \leftarrow;$$
$$Path((X \,.\, Z \,.\, \mathbf{nil}) \,.\, U, X, Y) \leftarrow Arc(X \,.\, Z \,.\, \mathbf{nil}), Path(U, Z, Y);$$

За счет правильного упорядочения атомов и программных утверждений программа $\mathcal{P}_2$ проводит вычисление маршрута более эффективно чем программа $\mathcal{P}_1$. На самом деле, обе программы несовершенны, поскольку обе могут зациклиться даже в случае простых графов. (Привести пример.)

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Но иногда и этих средств управления вычислением программ недостаточно для эффективного решения задачи.

Предположим, что нам нужно написать программу, которая проверяет, верно ли, что заданная буква содержится в заданном слове (например, буква a в слове $abaa$).

Можно предложить вот такую программу:

$$G: \quad ?Elem(a, a \centerdot b \centerdot a \centerdot a \centerdot \mathbf{nil})$$

$$\begin{aligned} \mathcal{P}: \quad & Elem(X, X \centerdot Y) \leftarrow; \\ & Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y); \end{aligned}$$

Тогда вычисления будут развиваться так.

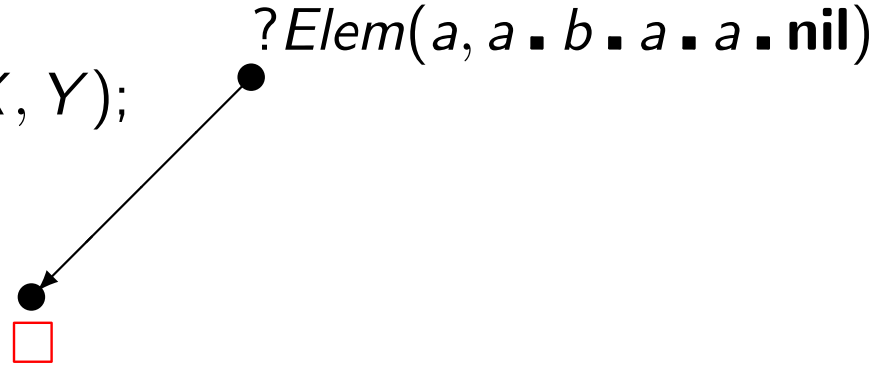
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \,.\, Y) \leftarrow$;
$Elem(X, Z \,.\, Y) \leftarrow Elem(X, Y)$;

$?Elem(a, a \,.\, b \,.\, a \,.\, a \,.\, \mathbf{nil})$

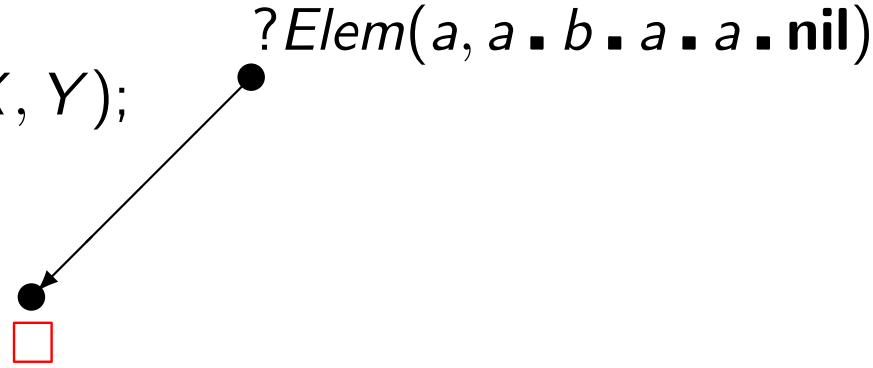
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;

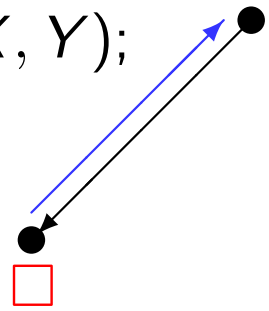


Ответ: YES

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

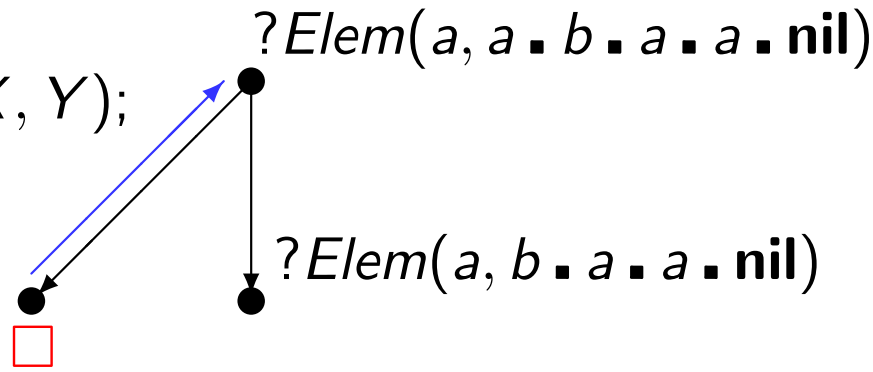
$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;

$?Elem(a, a \centerdot b \centerdot a \centerdot a \centerdot \mathbf{nil})$



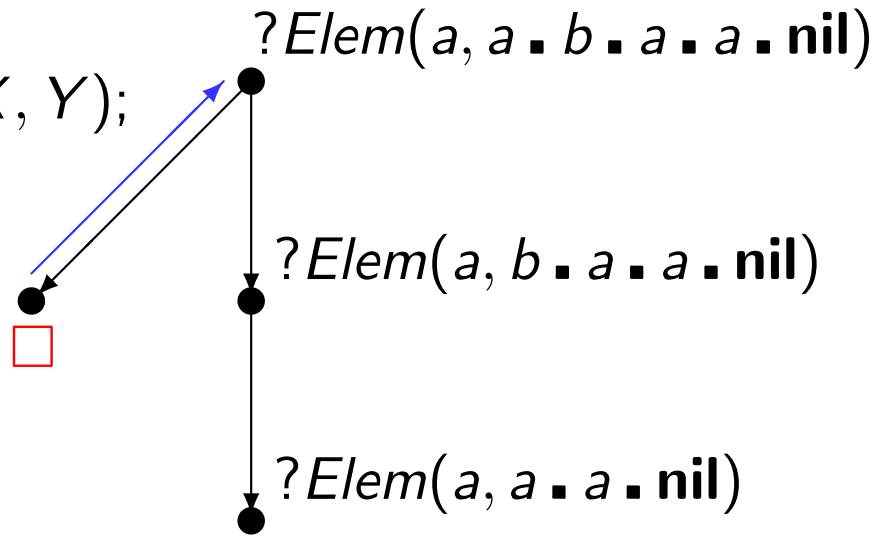
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \,.\, Y) \leftarrow$;
$Elem(X, Z \,.\, Y) \leftarrow Elem(X, Y)$;



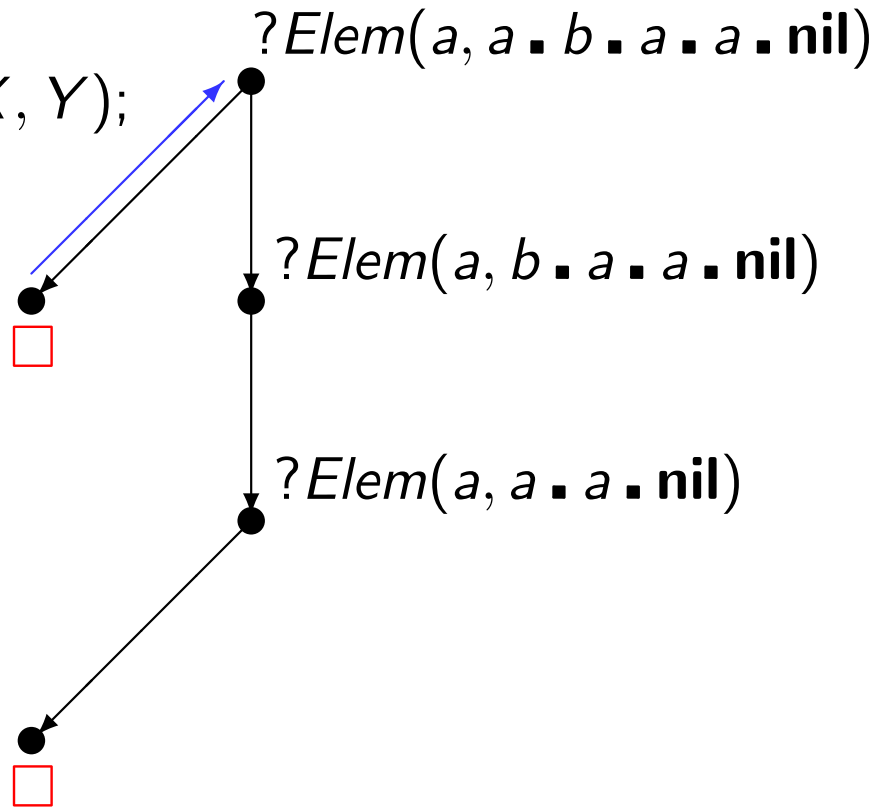
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \,.\, Y) \leftarrow$;
$Elem(X, Z \,.\, Y) \leftarrow Elem(X, Y)$;



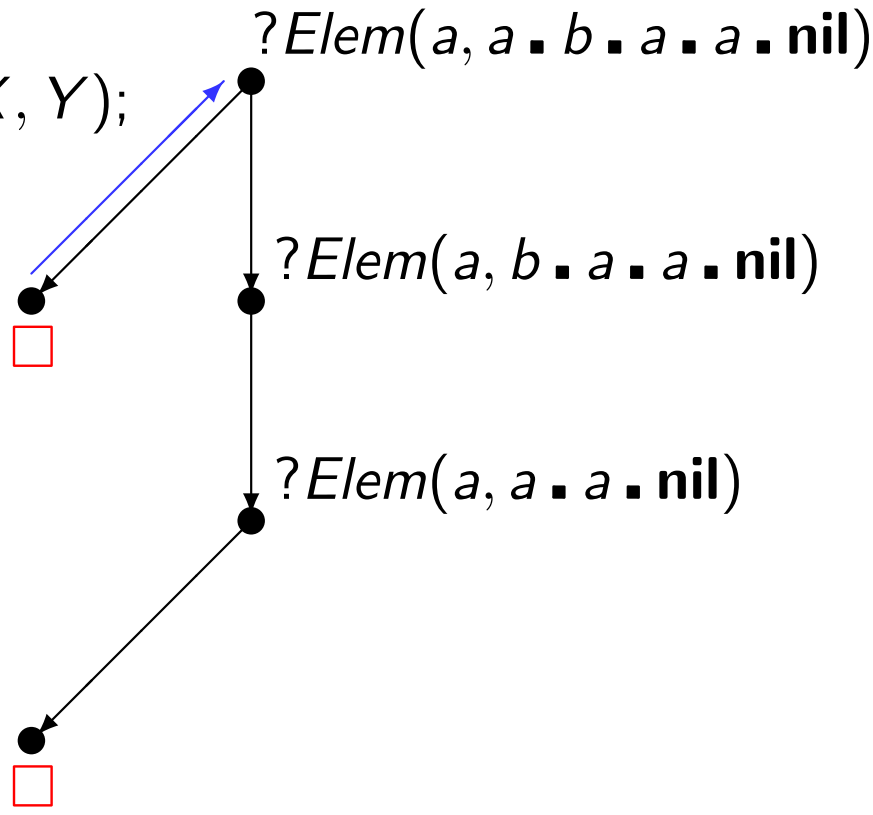
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

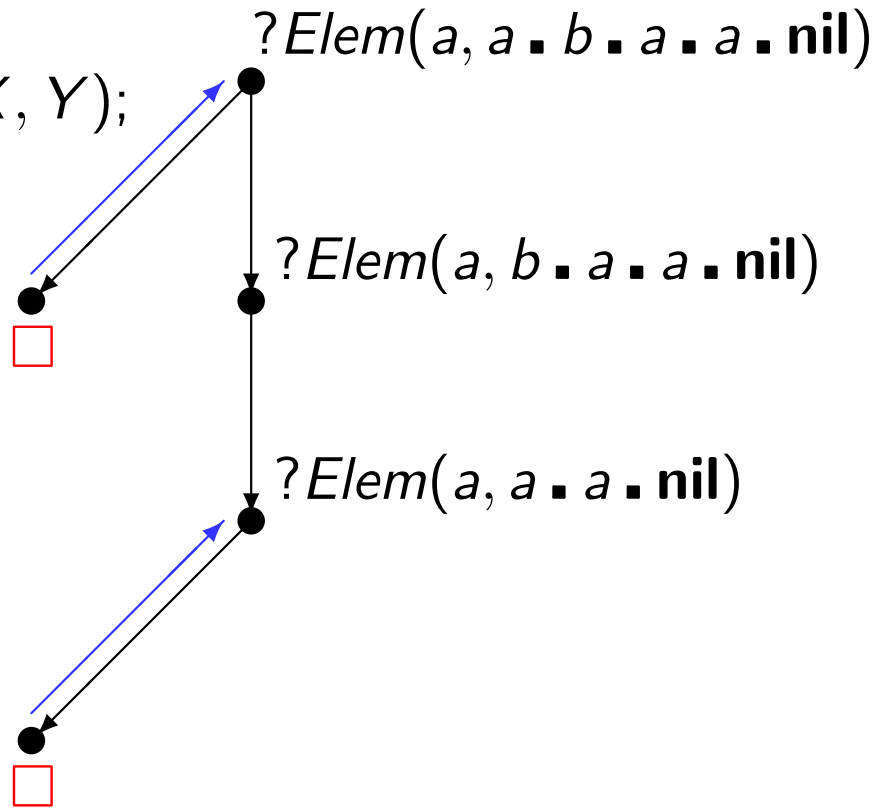
$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



Ответ: YES

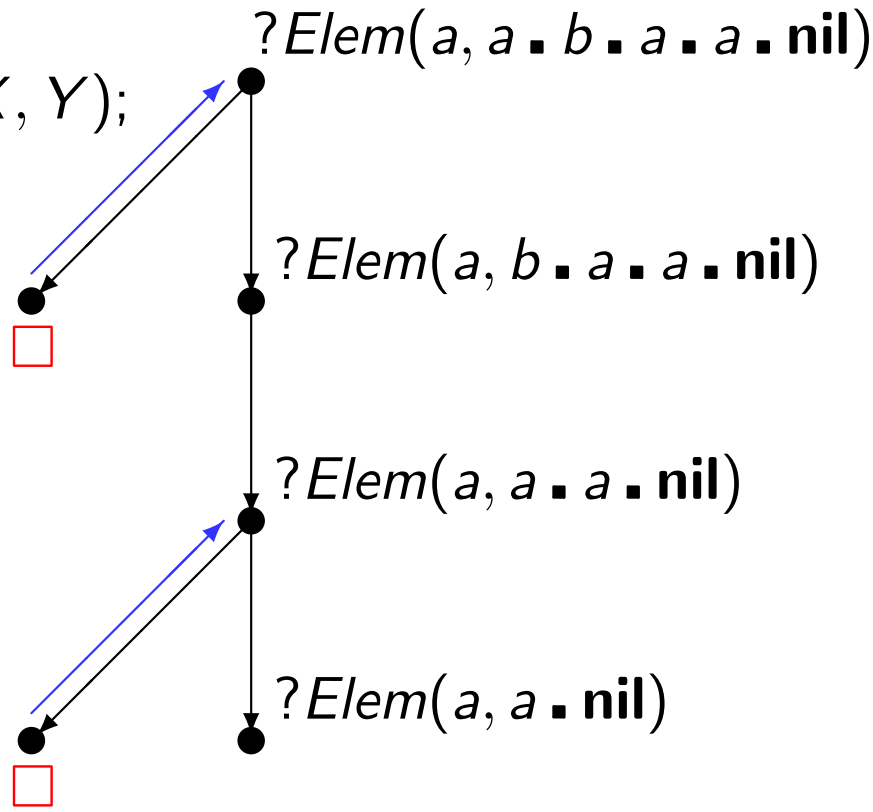
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



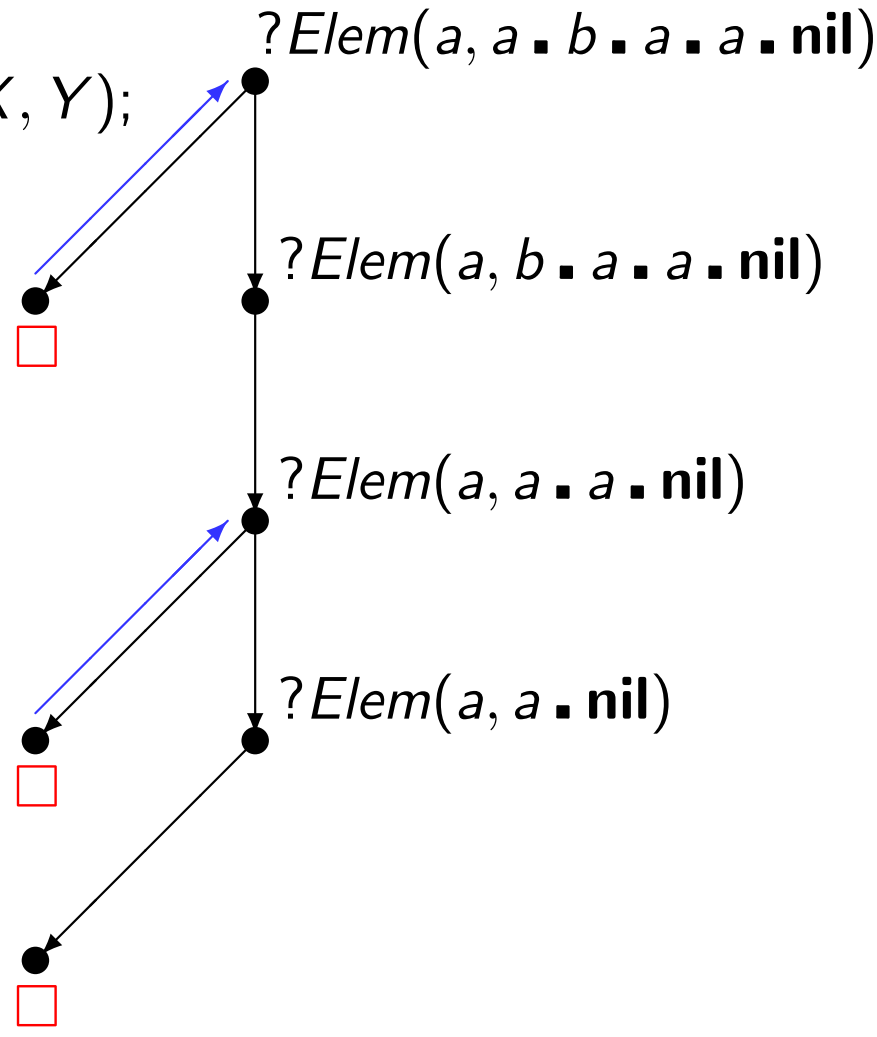
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



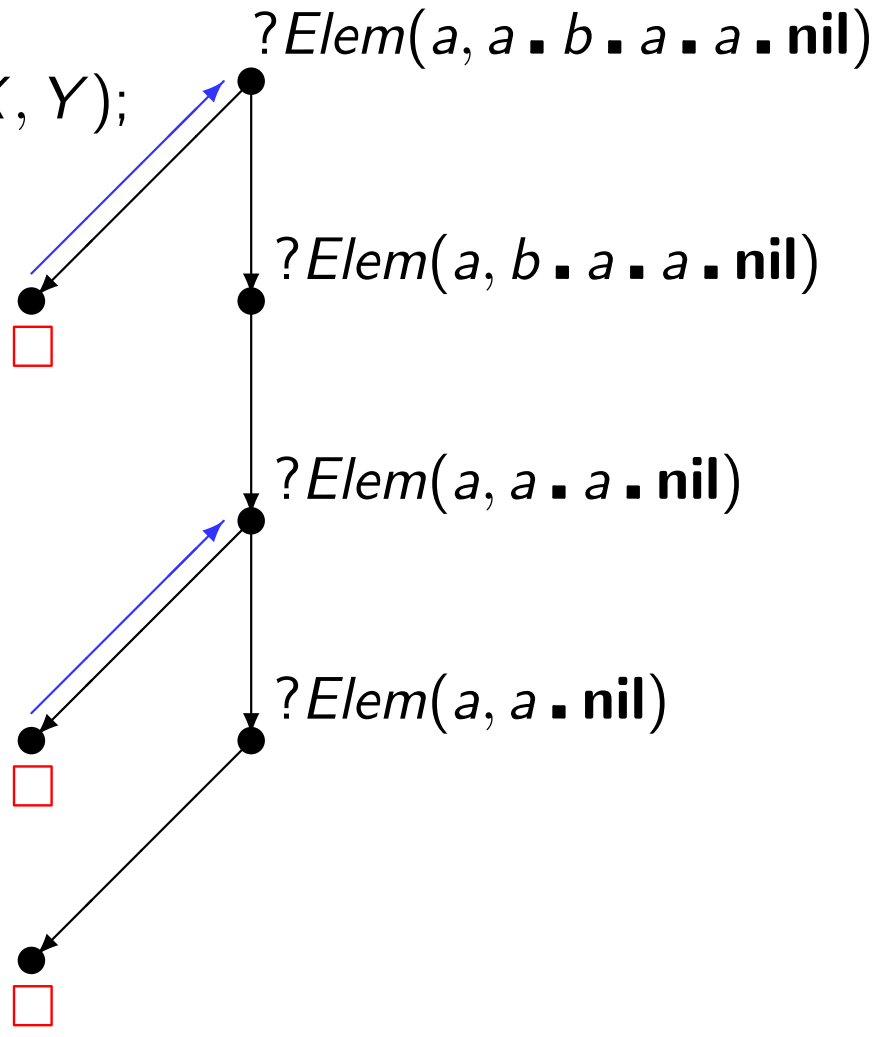
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

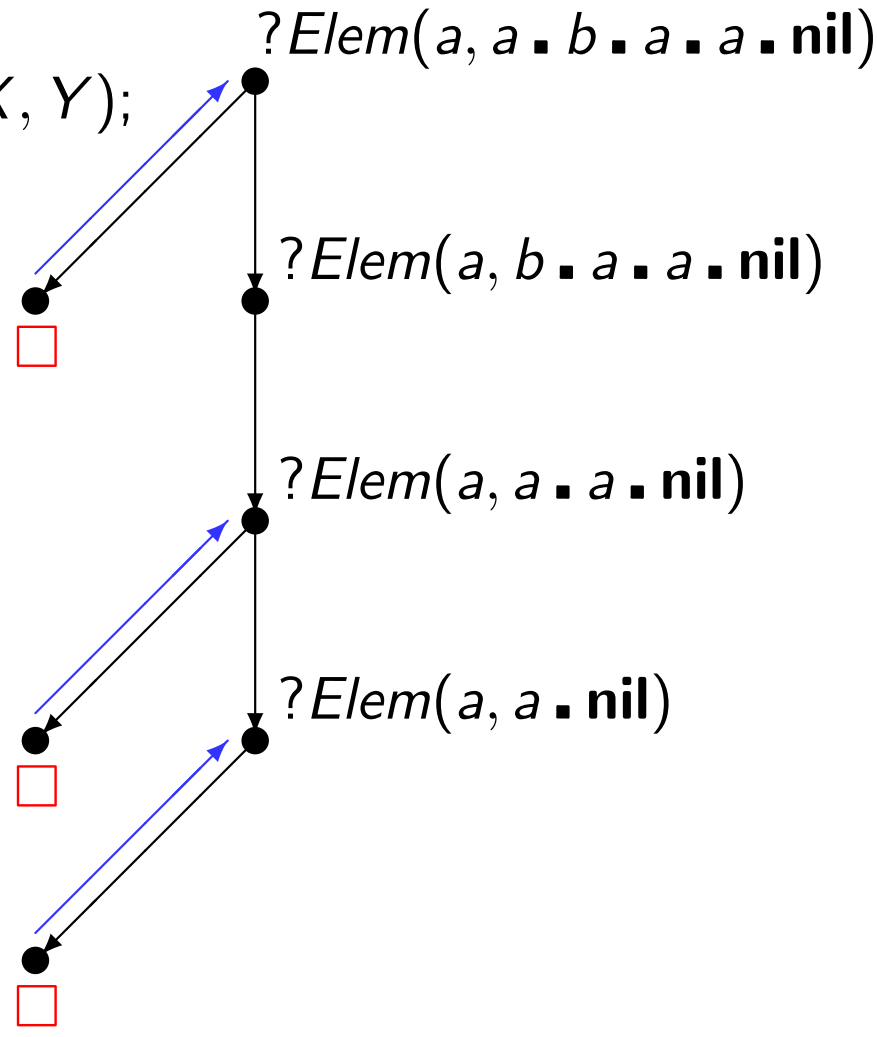
$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



Ответ: YES

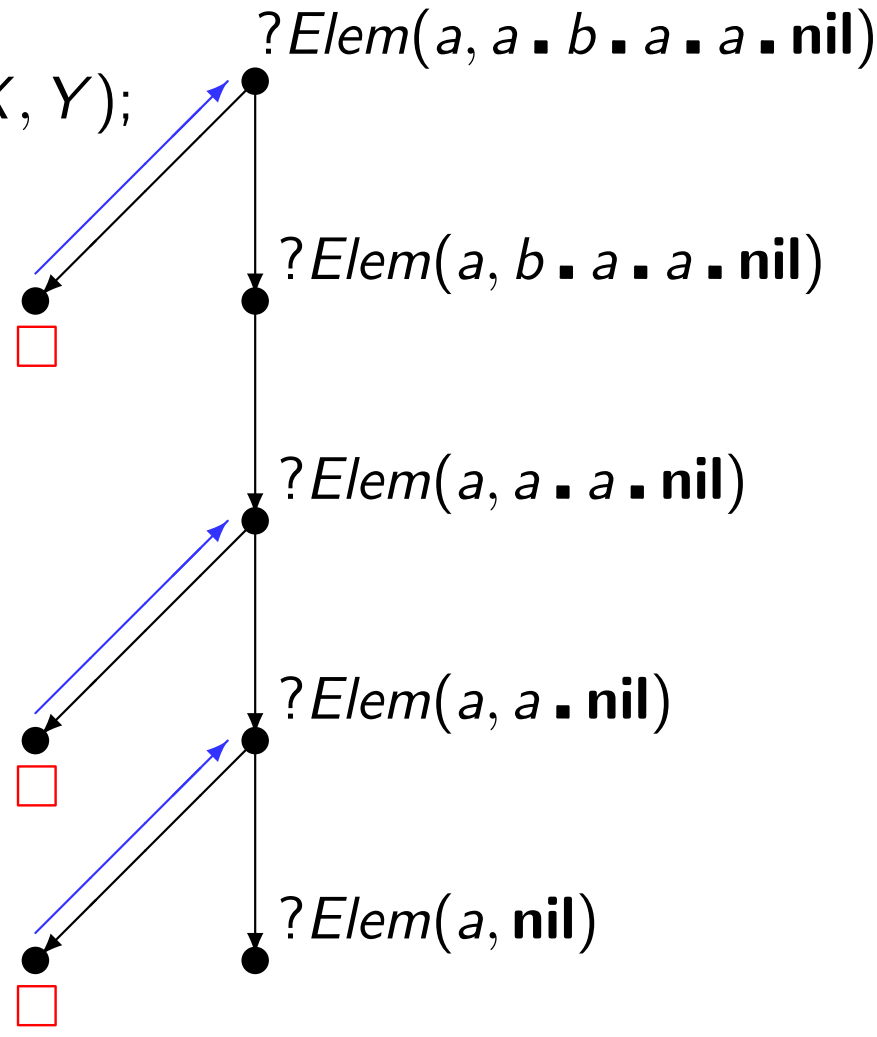
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



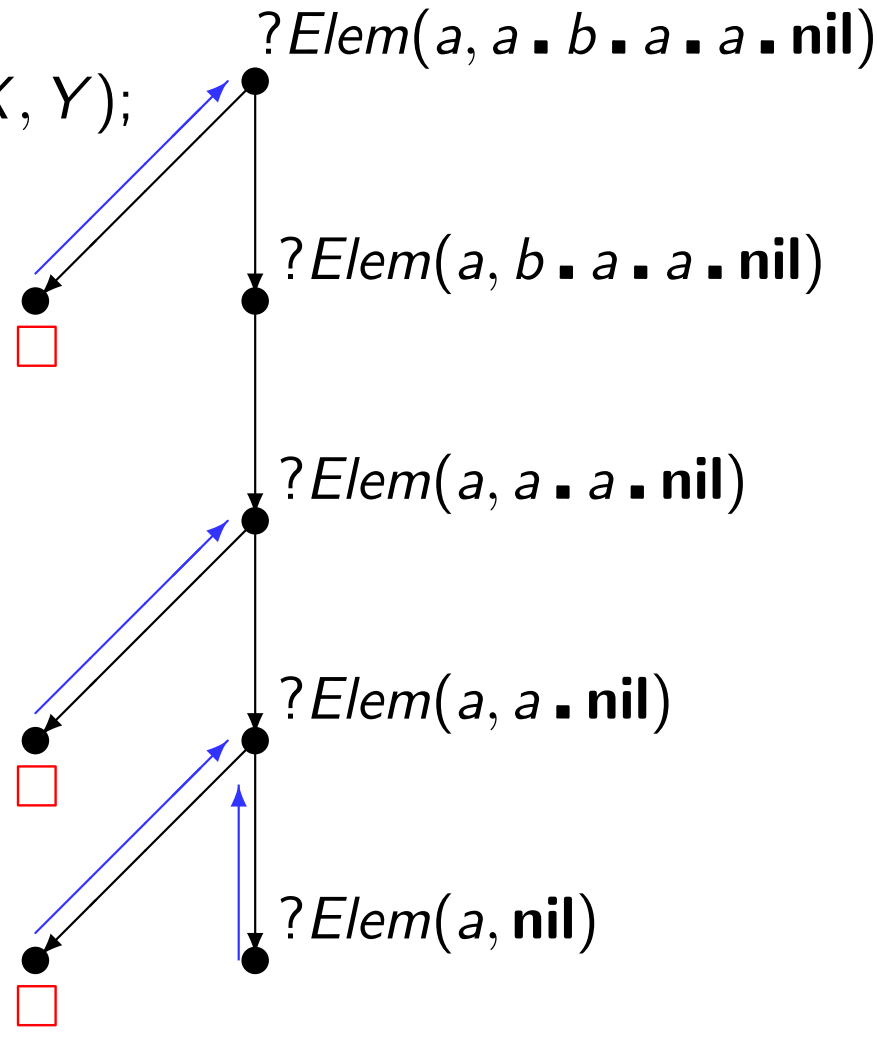
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



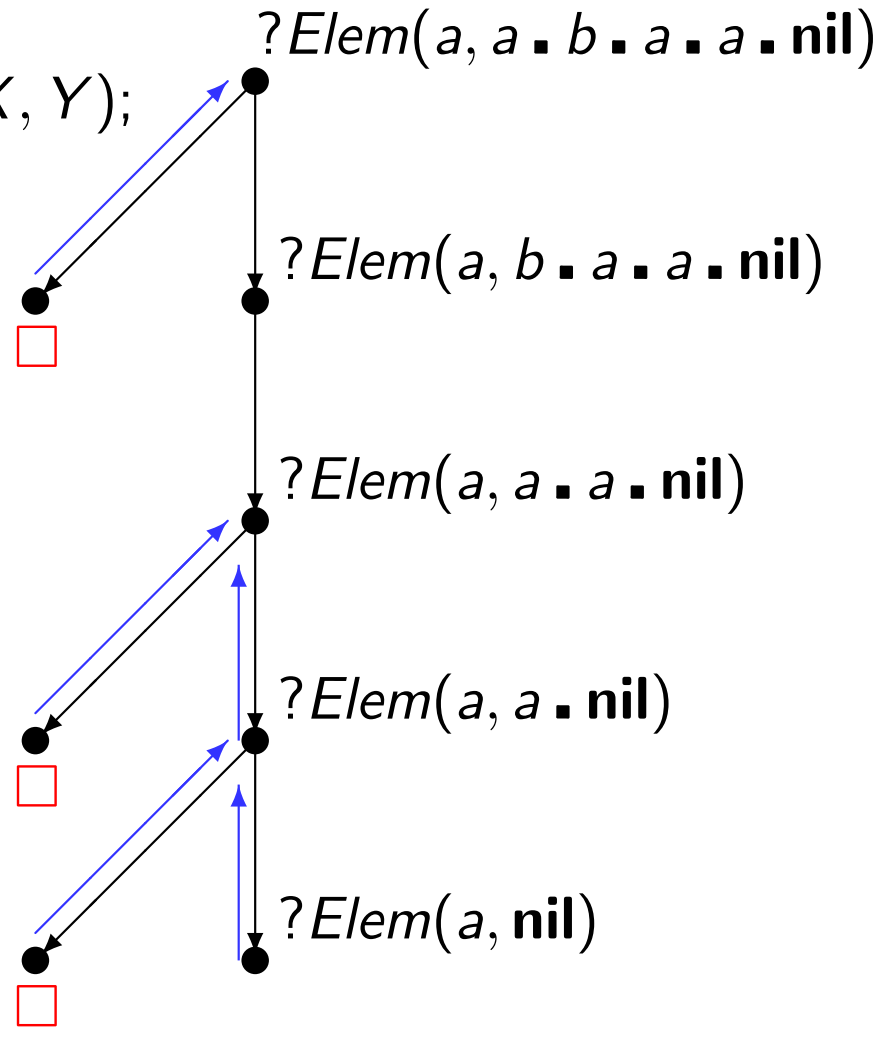
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



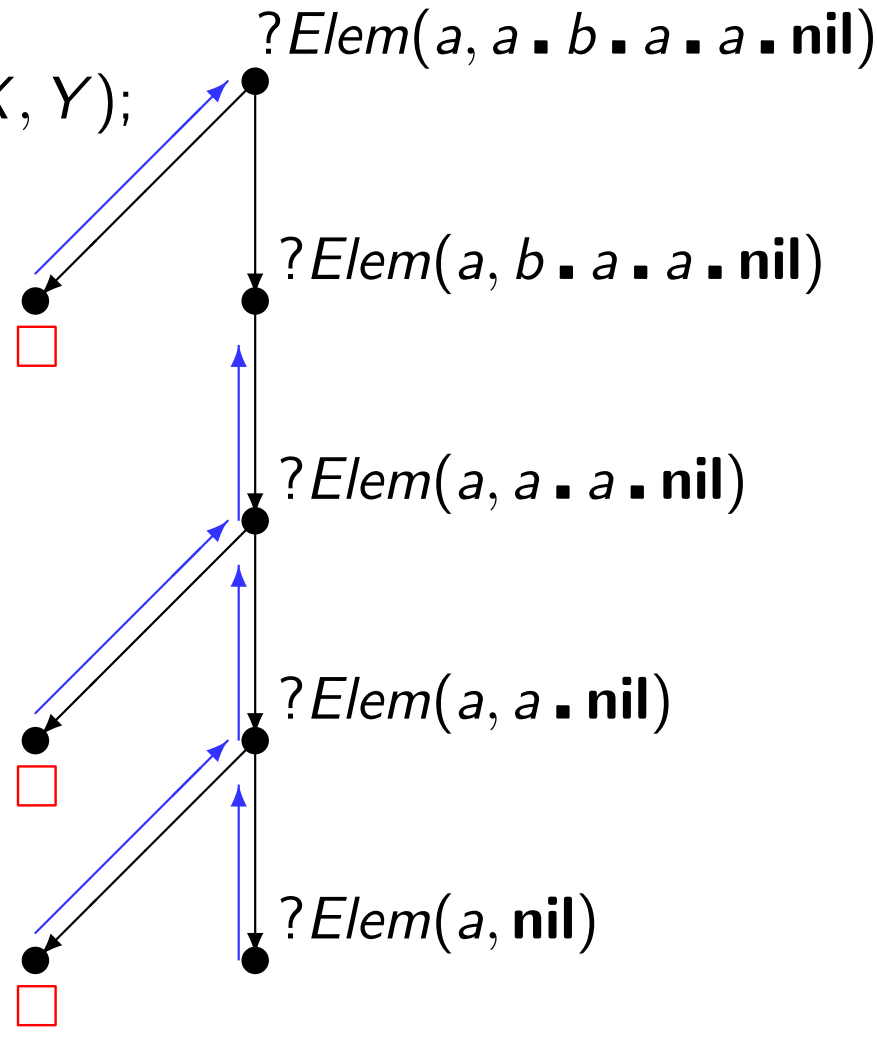
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



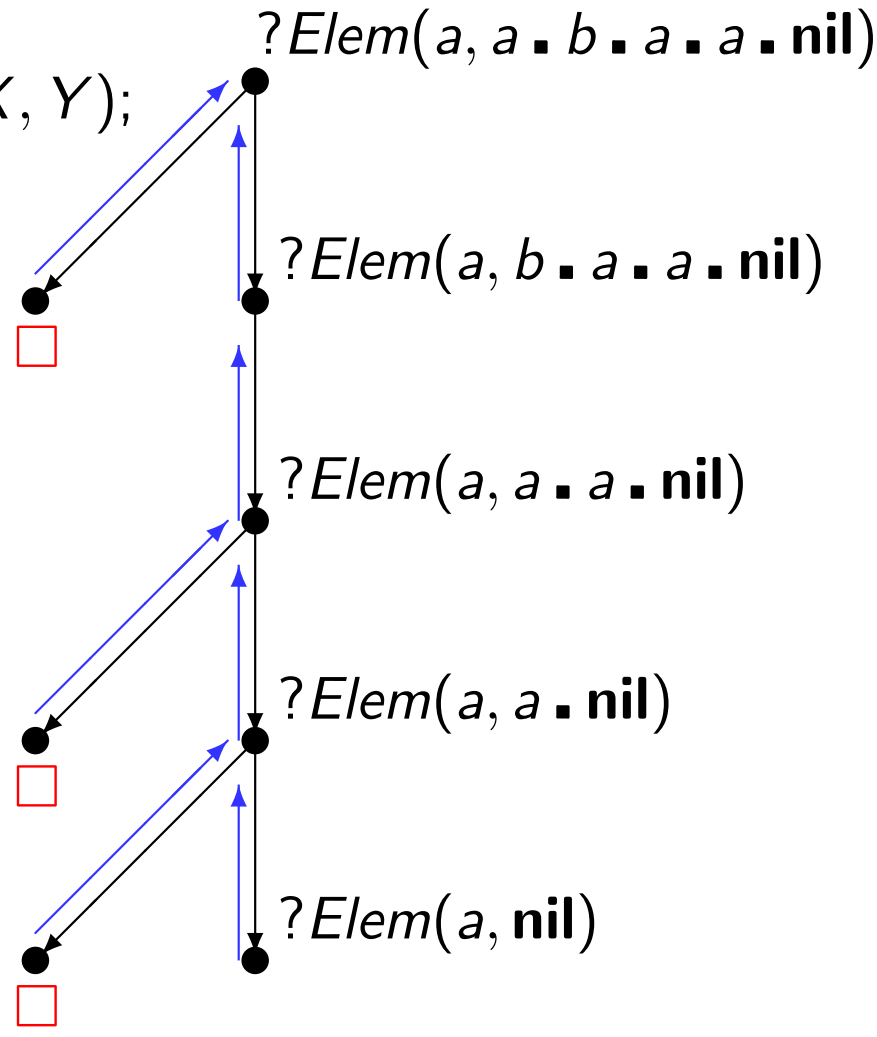
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



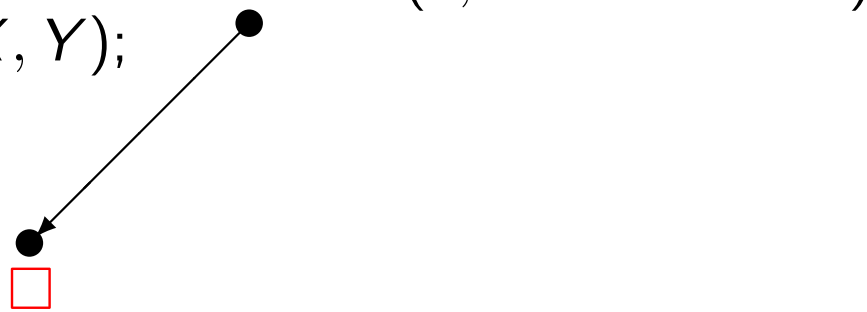
УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;



УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;

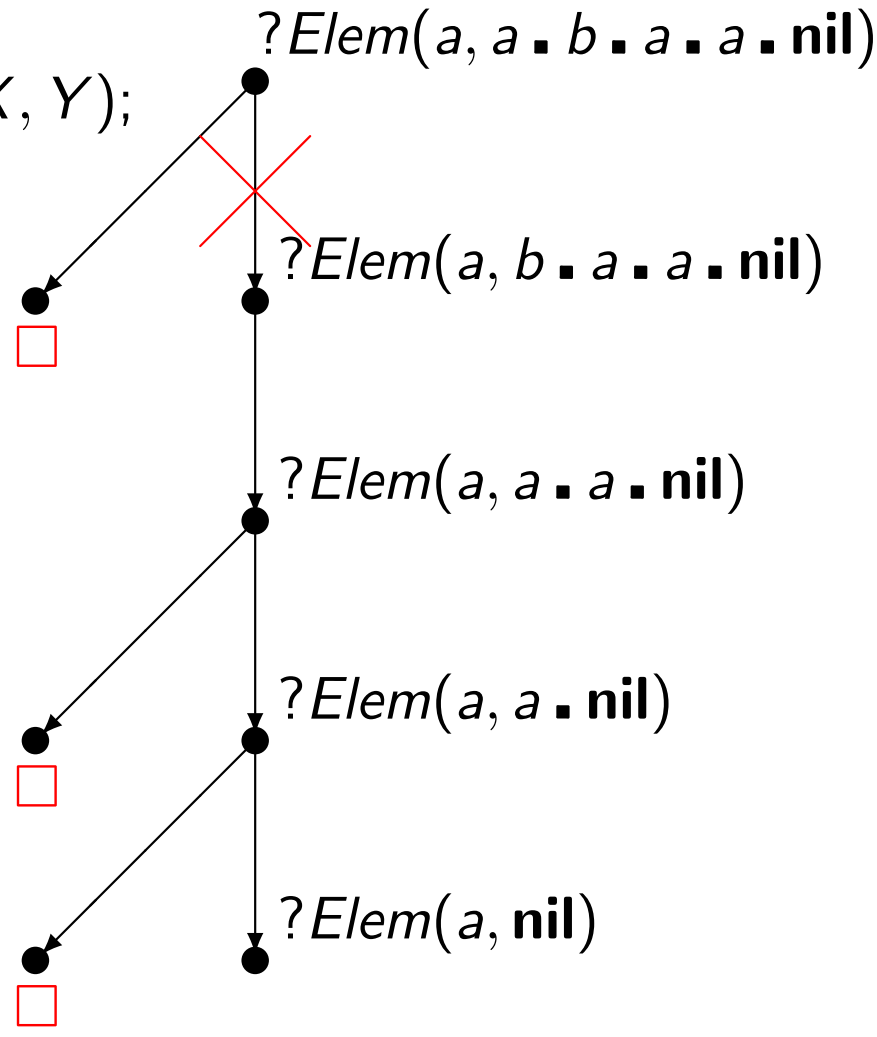


Но зачем нам обходить все дерево,
если для ответа YES достаточно
пройти по одной ветви?

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

$\mathcal{P}$: $Elem(X, X \centerdot Y) \leftarrow$;
$Elem(X, Z \centerdot Y) \leftarrow Elem(X, Y)$;

Хотелось бы иметь способ обрезать в дереве вычислений ненужные ветви.



ОПЕРАТОР ОТСЕЧЕНИЯ

Для этого в языках логического программирования вводится **оператор отсечения** (**cut**).

Этот оператор предсталяет собой 0-местный предикат **!**, оказывающий специальный побочный эффект.

С точки зрения декларативной семантики, предикат **!** имеет постоянное значение **true**. Для его описания не требуется никаких программных утверждений (**!** — встроенный предикат). Поэтому оператор отсечения может использоваться в запросах и в телах программных утверждений, не оказывая при этом никакого влияния на их логический смысл.

Однако операционная семантика оператора **!** определяется вне рамок SLD-резолютивного вывода. Оператор отсечения предназначен для выделения тех ветвей в дереве SLD-резолютивных вычислений, которые **не должны** проходиться по ходу вычисления запроса.

ОПЕРАТОР ОТСЕЧЕНИЯ

Операционная семантика оператора отсечения $!$ задается следующими правилами:

- Если запрос G и программное утверждение $D : A_0 \leftarrow A_1, \ldots, !, \ldots, A_n$ порождают SLD-резольвенту G', то в стеке вычислений программы запрос G получает специальную служебную пометку $(*$, индивидуальную для каждого вхождения оператора $!$;
- Если в запросе G оператор $!$ активен, т. е. $G = ?!, C_1, \ldots, C_k$, то в стеке вычислений программы запрос G получает специальную служебную пометку $*)$, индивидуальную для каждого вхождения оператора $!$, и при этом порождается новый запрос $G' = ?C_1, \ldots, C_k$;
- Если по ходу вычисления **при откате** достигается элемент стека вычислений программы, помеченный $*)$, то из стека удаляются все элементы, расположенные между элементами, помеченными $(*$ и $*)$ (включая и сами эти элементы).

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), \mathbf{!}, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

$?P(U, V), R(U)$	
$\eta = \varepsilon; \quad count = 1$	

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), \mathbf{!}, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 1$	

Протокол:

SLD-резолюция

Появление оператора $!$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 1$	

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 2$	

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	

Протокол:

SLD-резолюция

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 1$	

Протокол:

Активизация оператора $!$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 1$	

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 2$	

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 3$	

Протокол:

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

SLD-резолюция

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 1$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 1$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 2$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Унификация

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

SLD-резолюция

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$	
$\square$ $\eta = \{U/b, V/c\};$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Успех: $\eta = \{U/b, V/c\}$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$	
$\square$ $\eta = \{U/b, V/c\};$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 3$	

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 4$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 3$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	
$?R(b)$ $\eta = \{U/b, V/c\}; count = 5$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 4$	

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	

Протокол:

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

SLD-резолюция

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 1$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 1$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 2$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 3$	

Унификация

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

SLD-резолюция

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 3$	
$\square$ $\eta = \{U/b, V/b\};$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Успех: $\eta = \{U/b, V/b\}$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 3$	
$\square$ $\eta = \{U/b, V/b\};$	

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 3$	

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 4$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	
$?R(b)$ $\eta = \{U/b, V/b\}; count = 5$	

Нет унификатора

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$
$?Q(V), R(b)$ $\eta = \{U/b\}; \quad count = 5$	

Протокол:

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	
$?!, Q(V), R(b)$ $\eta = \{U/b\};$	$*)$

Откат

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

$?P(U, V), R(U)$ $\eta = \varepsilon; \quad count = 1$	$(*$
$?R(U), !, Q(V), R(U)$ $\eta = \varepsilon; \quad count = 4$	

Протокол:

Срабатывание оператора $!$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

$?P(U, V), R(U)$	$($ *
$\eta = \varepsilon;\quad count = 1$	

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), \mathbf{!}, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Срабатывание оператора $\mathbf{!}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), \mathbf{!}, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Срабатывание оператора $\mathbf{!}$

УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ

Пример вычисления логических программ с оператором отсечения

Запрос: $?P(U, V), R(U)$

Программа $\mathcal{P}$:

$$P(X, Y) \leftarrow R(X), \mathbf{!}, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$

Протокол:

Конец вычислений

Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

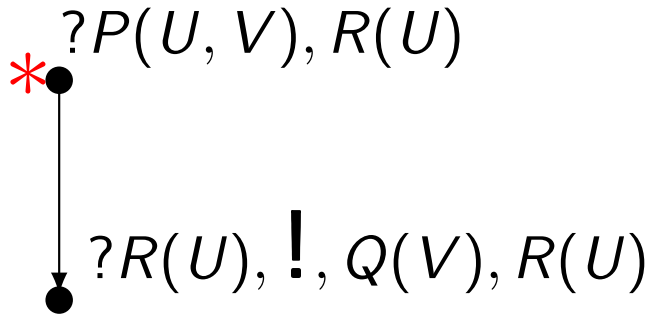
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)

$?P(U, V), R(U)$

Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

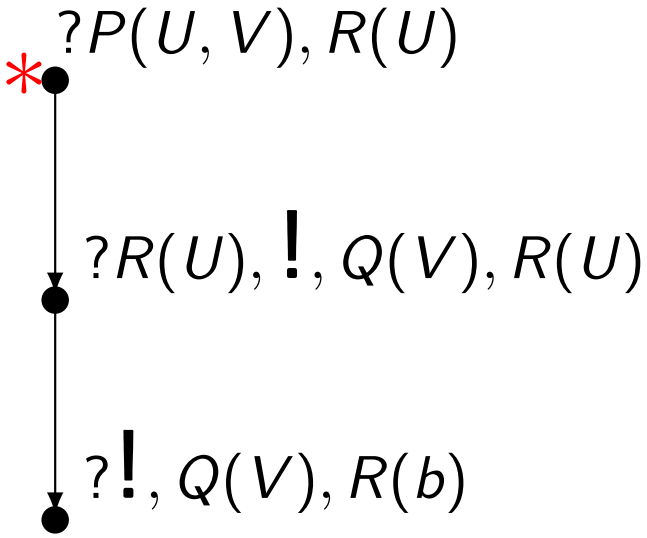
$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

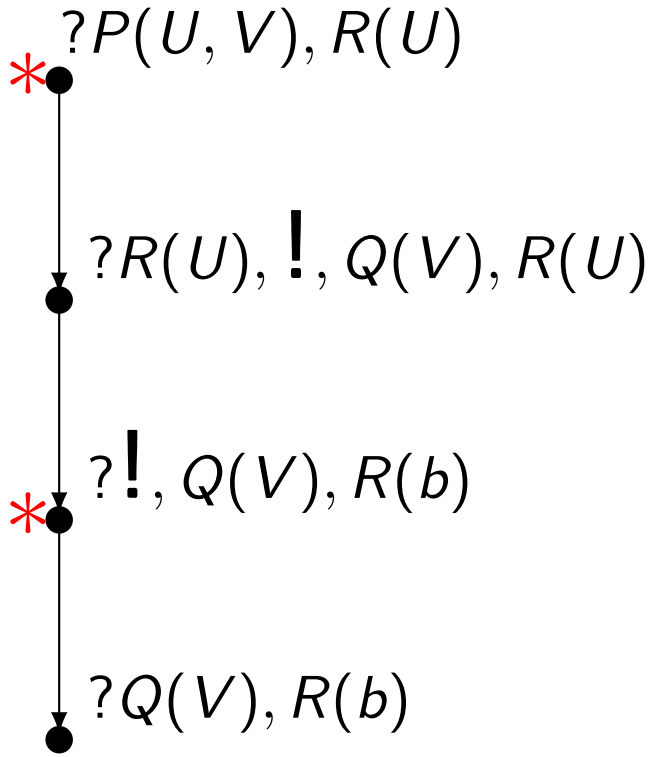
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

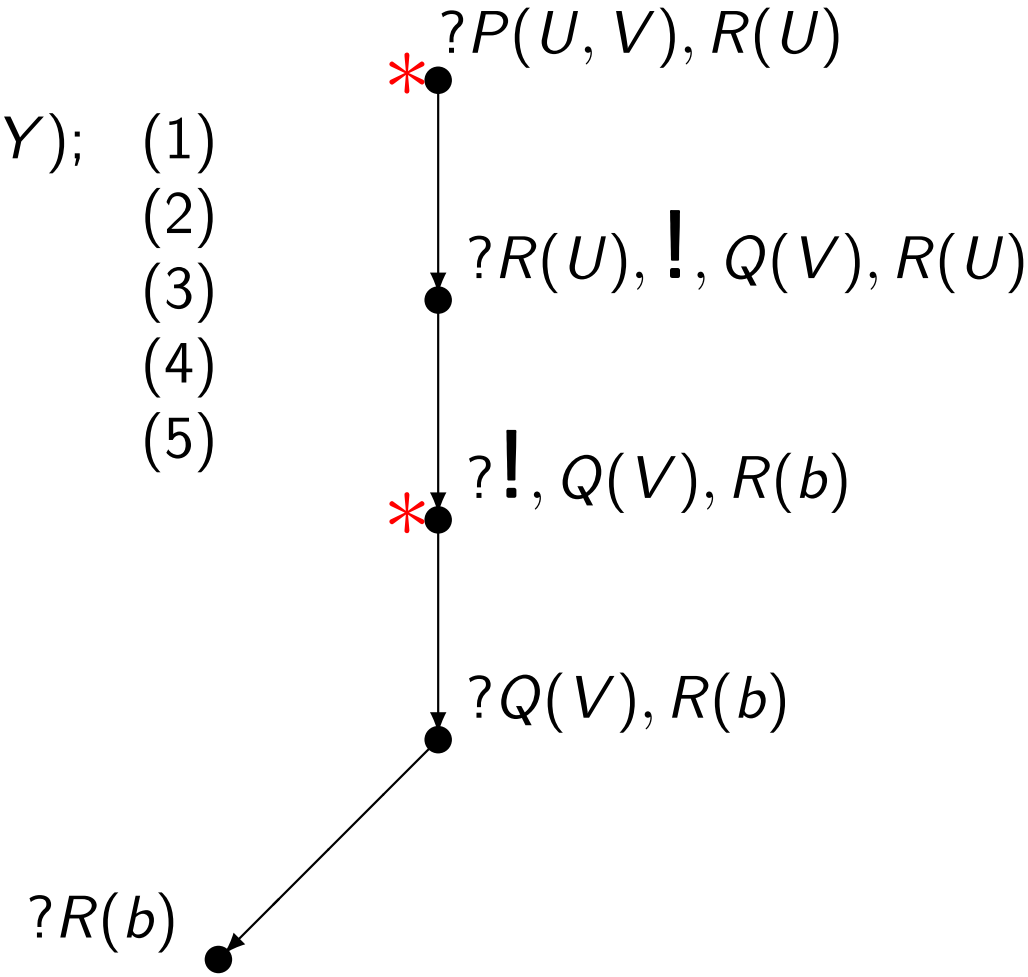
$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

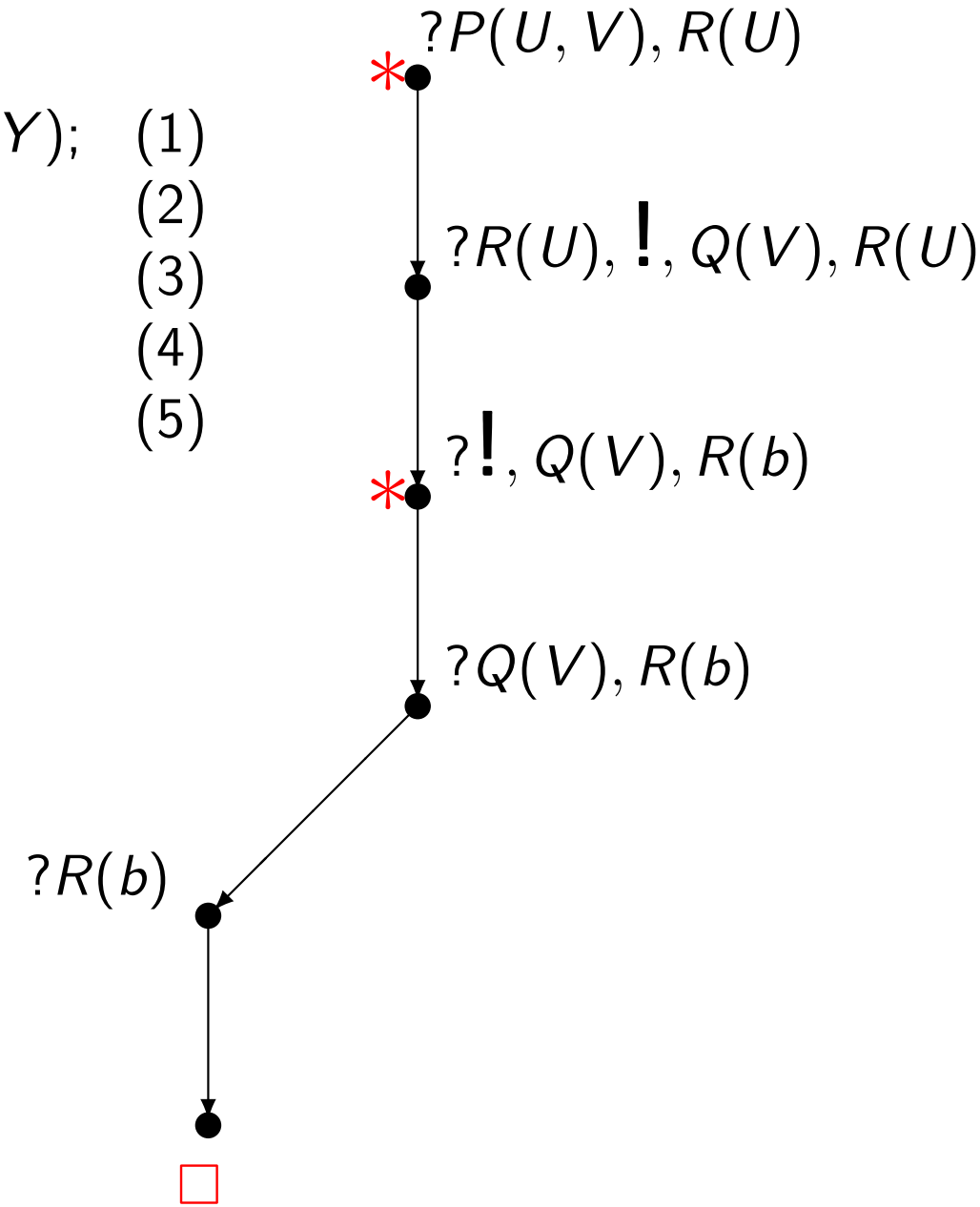
$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

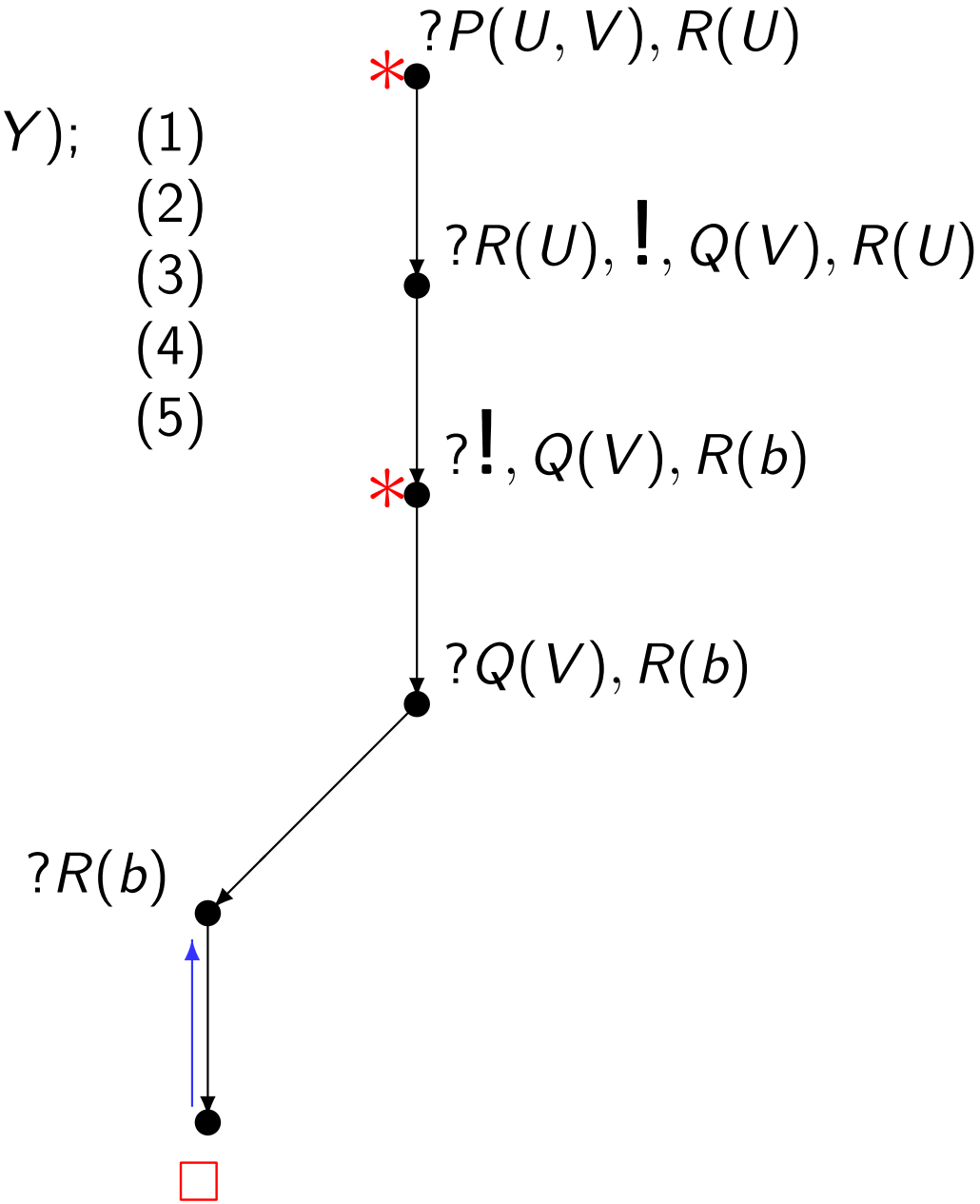
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

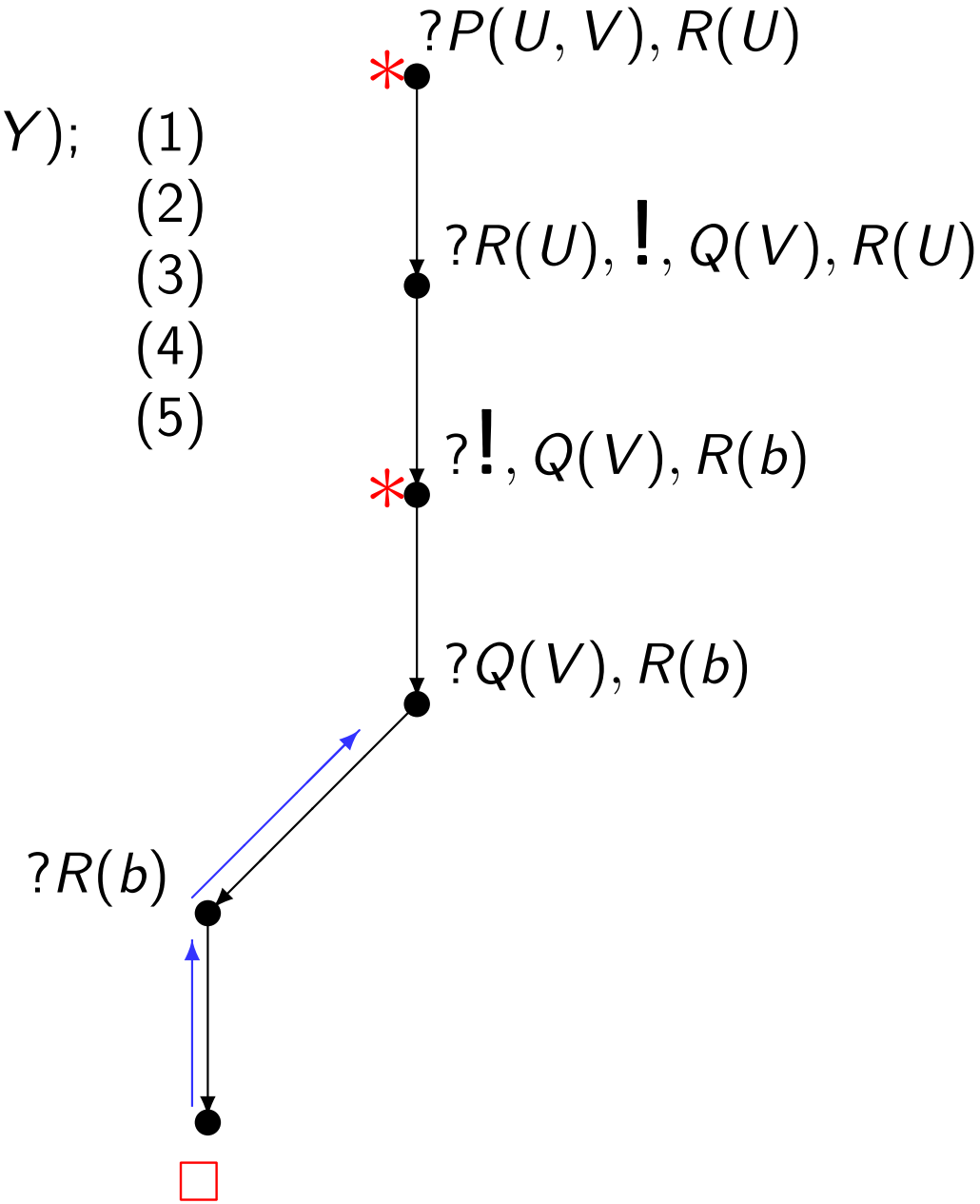
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

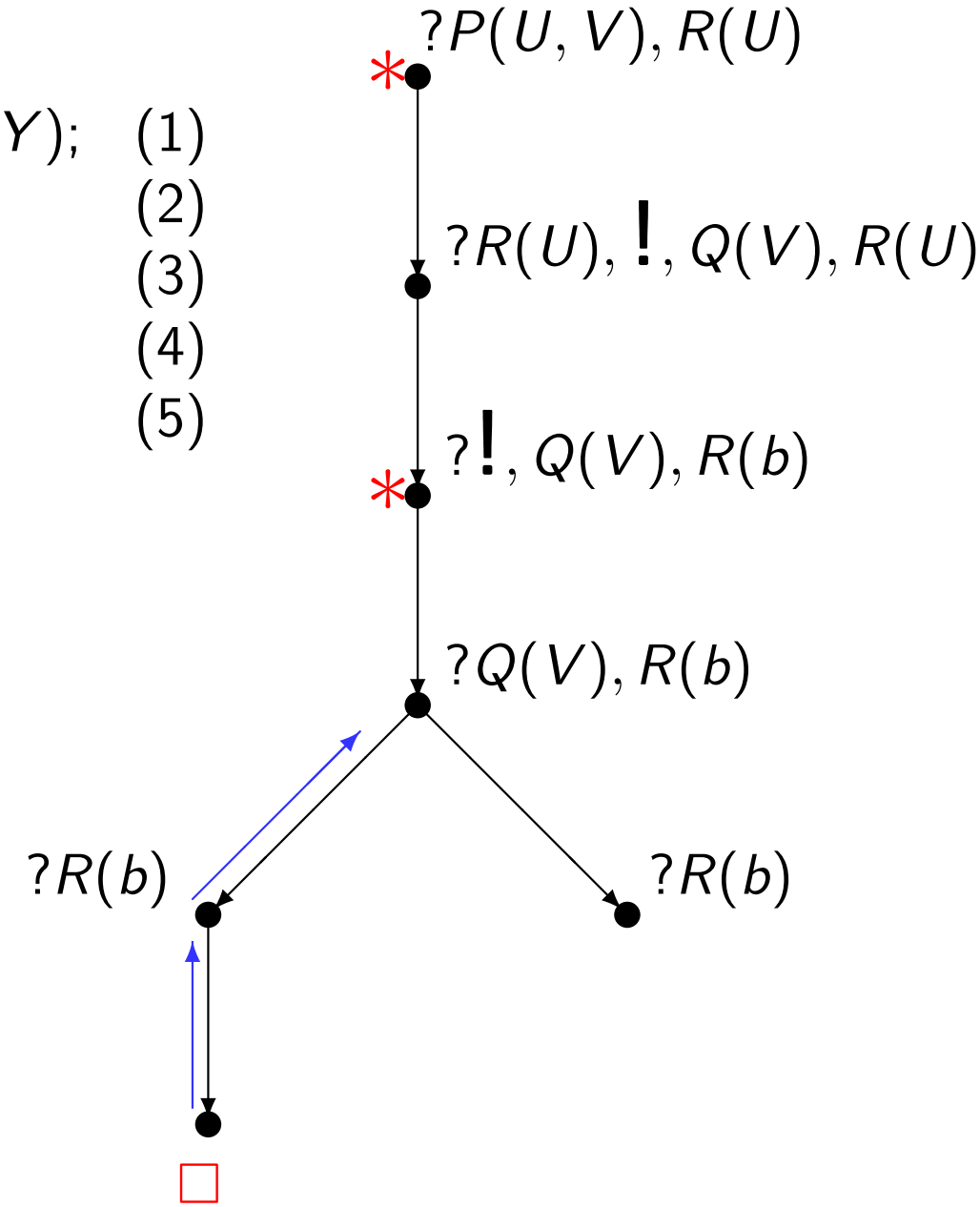
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

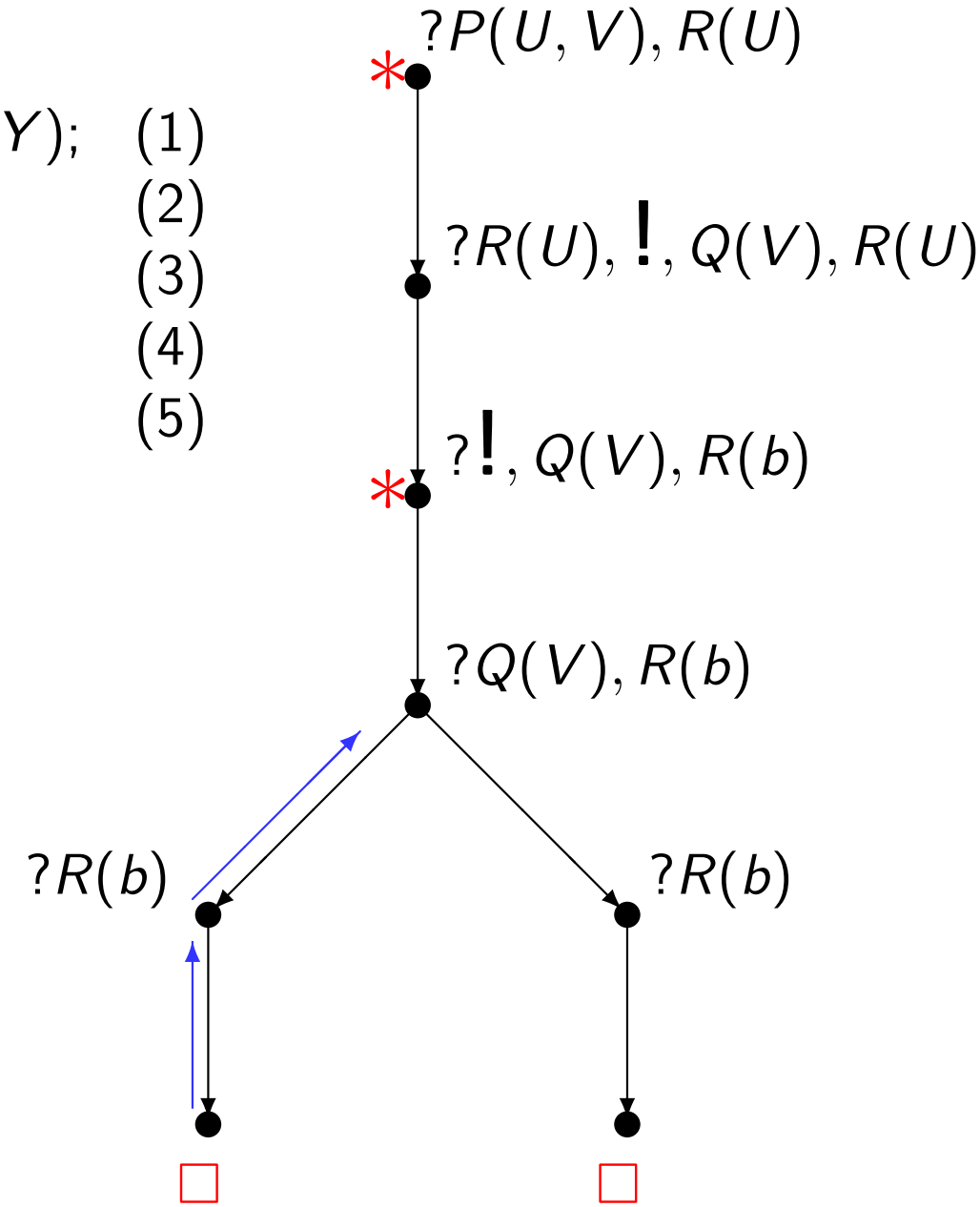
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

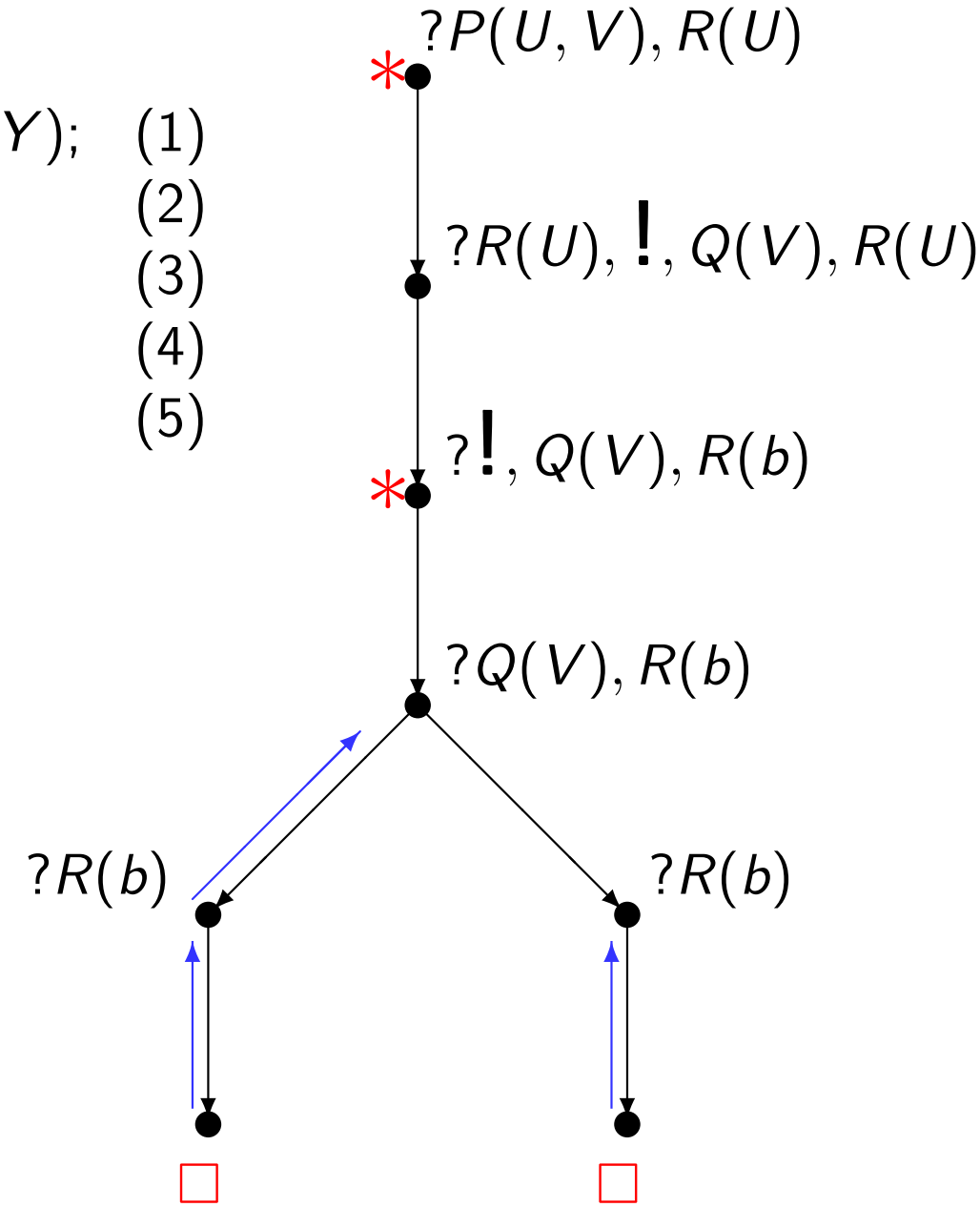
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

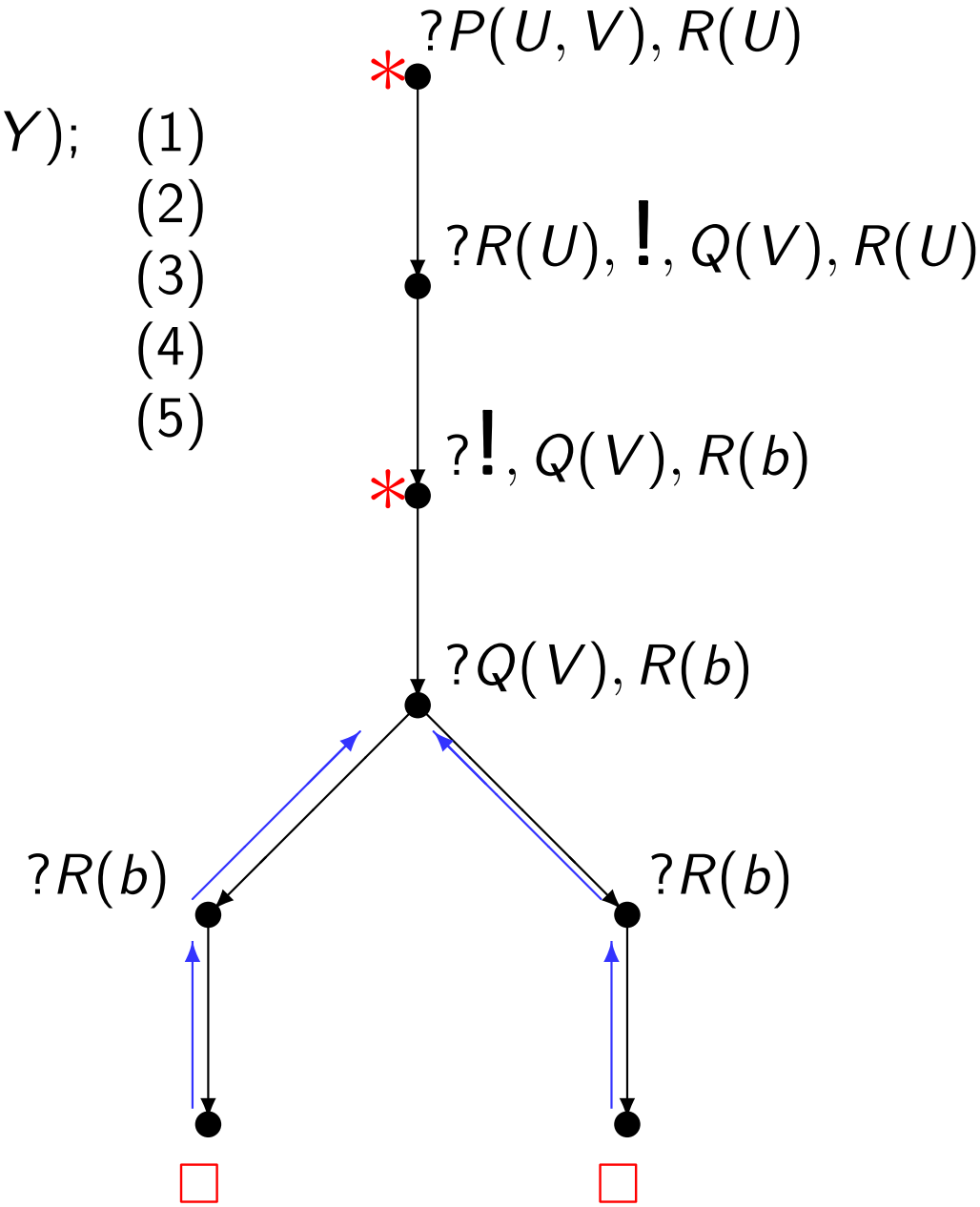
$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

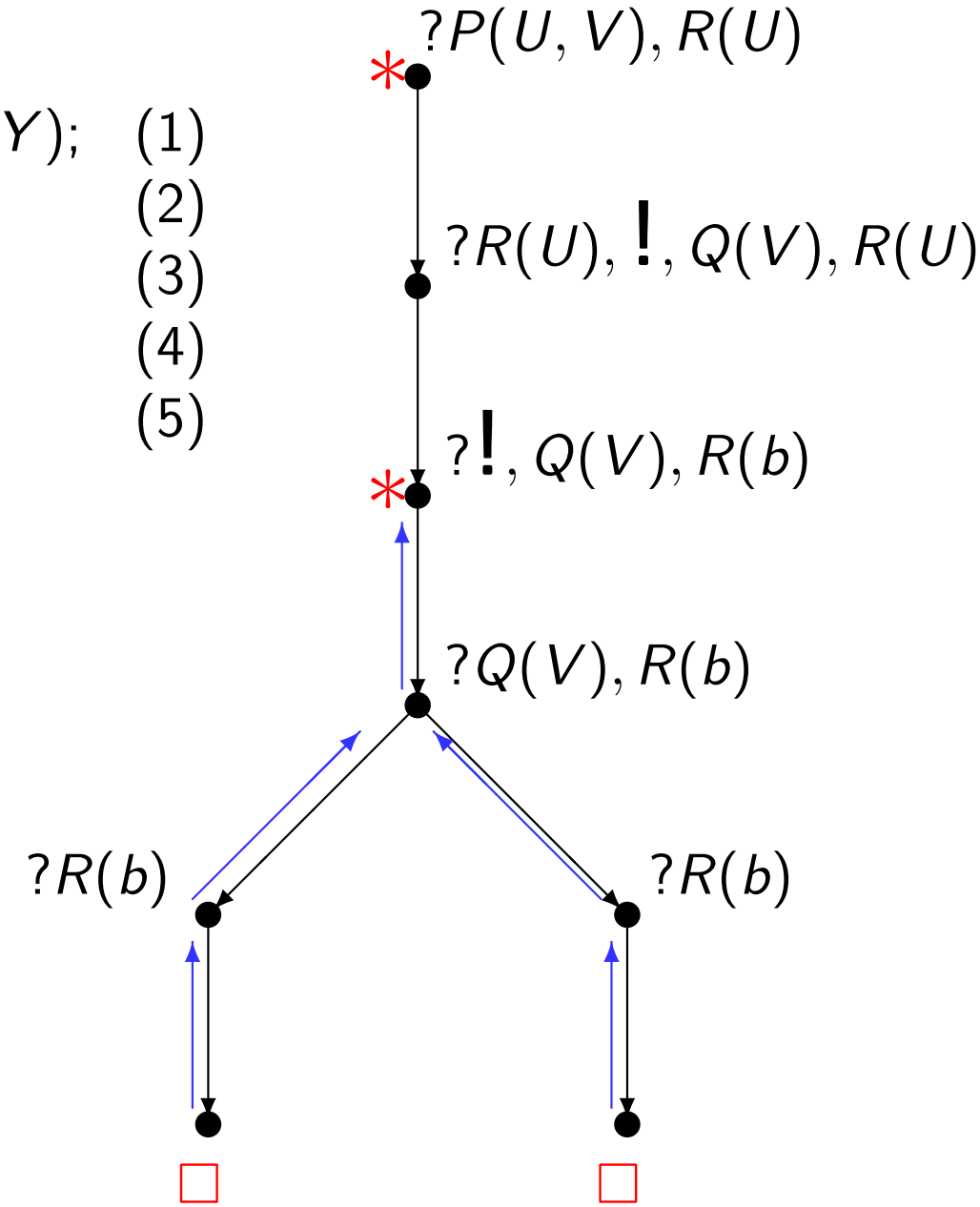
$$P(X, Y) \leftarrow R(X), !, Q(Y); \quad (1)$$
$$P(X, X) \leftarrow Q(X); \quad (2)$$
$$R(b) \leftarrow; \quad (3)$$
$$Q(c) \leftarrow; \quad (4)$$
$$Q(b) \leftarrow; \quad (5)$$



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

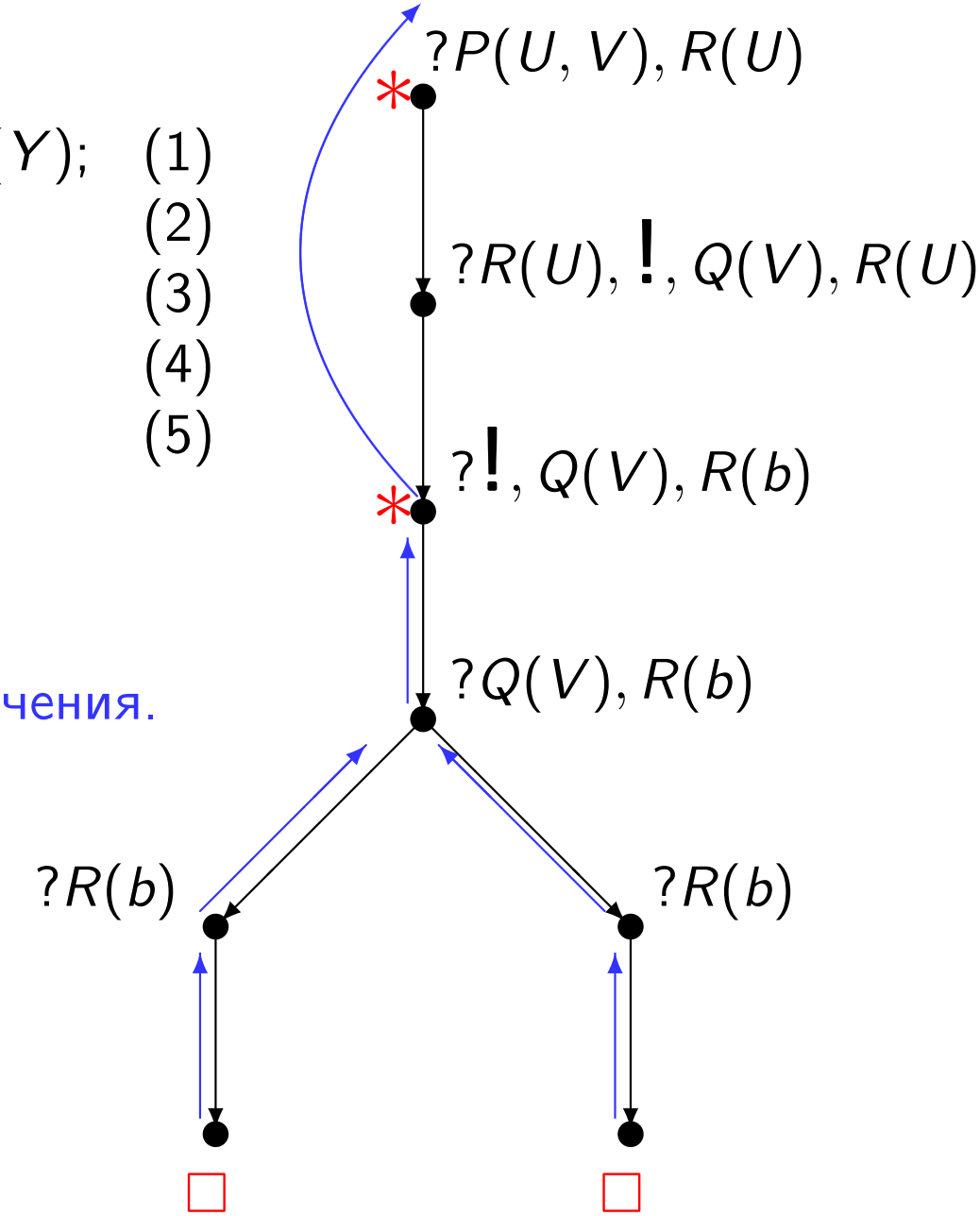
$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



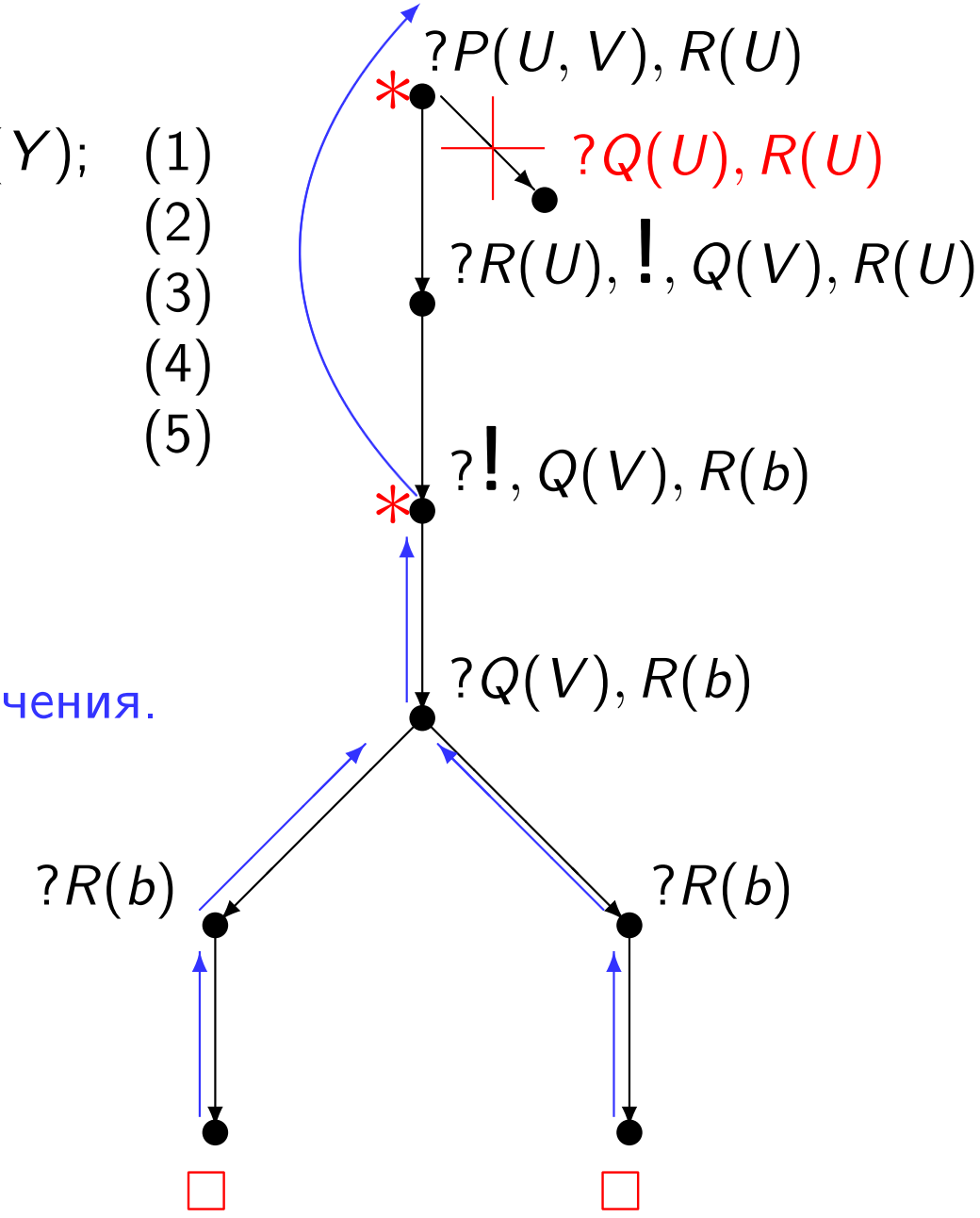
Сработал оператор отсечения.

Построение дерева вычислений завершено.

Дерево SLD-резолютивных вычислений

Программа $\mathcal{P}$:

$P(X, Y) \leftarrow R(X), !, Q(Y);$ (1)
$P(X, X) \leftarrow Q(X);$ (2)
$R(b) \leftarrow;$ (3)
$Q(c) \leftarrow;$ (4)
$Q(b) \leftarrow;$ (5)



Сработал оператор отсечения.

Построение дерева вычислений завершено.

При этом часть ветвей была отсечена.

ОПЕРАТОР ОТСЕЧЕНИЯ

Программное утверждение

$$A_0 \leftarrow A_1, \ldots, A_k, !, A_{k+1}, \ldots, A_n;$$

содержащее оператор отсечения можно прочитывать двояко:

- Чтобы решить задачу A_0 нужно найти **только первое** решение задач $A_1, \ldots, A_k$ и далее решать задачи $A_{k+1}, \ldots, A_n$. Если решение задач $A_1, \ldots, A_k$ найти не удается, то воспользоваться альтернативными процедурами решения задачи A_0.
- Чтобы решить задачу A_0 нужно проверить условия $A_1, \ldots, A_k$. Если эти условия выполнены, то приступить к решению задач $A_{k+1}, \ldots, A_n$ и не обращаться к другим вариантам решения задачи A_0. Если же эти условия не выполнены, то обратиться к альтернативным способам решения задачи A_0.

ОПЕРАТОР ОТСЕЧЕНИЯ

Таким образом, оператор отсечения позволяет удобно использовать в логическом программировании стандартные конструкции императивного программирования.

- **Ветвление.** S_0: **if** P **then** S_1 **else** S_2 **fi**

$$\mathcal{P}_{if-then-else}: \quad \begin{array}{l} S_0 \leftarrow P, !, S_1; \\ S_0 \leftarrow S_2; \end{array}$$

- **Итерация.** S_0: **while** P **do** S_1 **od**

$$\mathcal{P}_{while-do}: \quad \begin{array}{l} S_0 \leftarrow P, !, S_1, S_0; \\ S_0 \leftarrow; \end{array}$$

ОПЕРАТОР ОТСЕЧЕНИЯ

С введением в логические программы оператора отсечения теорема полноты операцинной семантики относительно декларативной семантики перестают быть справедливыми.

ОПЕРАТОР ОТСЕЧЕНИЯ

Программы $\mathcal{P}_1$ и $\mathcal{P}_2$ вычисления гласных букв

$\mathcal{P}_1$

$Elem\_Vow(X, X \centerdot Y) \leftarrow Vowel(X);$
$Elem\_Vow(X, Z \centerdot Y) \leftarrow Elem\_Vow(X, Y);$
$Vowel(a) \leftarrow;$
$Vowel(e) \leftarrow;$
$Vowel(i) \leftarrow;$
$Vowel(o) \leftarrow;$
$Vowel(u) \leftarrow;$
$Vowel(y) \leftarrow;$

$\mathcal{P}_2$

$Elem\_Vow(X, X \centerdot Y) \leftarrow Vowel(X), !;$
$Elem_Vow(X, Z \centerdot Y) \leftarrow Elem\_Vow(X, Y);$
$Vowel(a) \leftarrow;$
$Vowel(e) \leftarrow;$
$Vowel(i) \leftarrow;$
$Vowel(o) \leftarrow;$
$Vowel(u) \leftarrow;$
$Vowel(y) \leftarrow;$

равносильны в декларативной семантике , и запрос

$$G :? Elem\_Vow(X, o \centerdot n \centerdot e \centerdot \mathbf{nil})$$

для обеих программ имеет два правильных ответа $\theta_1 = \{X/o\}$ и $\theta_2 = \{X/e\}$.

ОПЕРАТОР ОТСЕЧЕНИЯ

Программы $\mathcal{P}_1$ и $\mathcal{P}_2$ вычисления гласных букв

$\mathcal{P}_1$

$Elem\_Vow(X, X \centerdot Y) \leftarrow Vowel(X);$
$Elem\_Vow(X, Z \centerdot Y) \leftarrow Elem\_Vow(X, Y);$
$Vowel(a) \leftarrow;$
$Vowel(e) \leftarrow;$
$Vowel(i) \leftarrow;$
$Vowel(o) \leftarrow;$
$Vowel(u) \leftarrow;$
$Vowel(y) \leftarrow;$

$\mathcal{P}_2$

$Elem\_Vow(X, X \centerdot Y) \leftarrow Vowel(X), !;$
$Elem_Vow(X, Z \centerdot Y) \leftarrow Elem\_Vow(X, Y);$
$Vowel(a) \leftarrow;$
$Vowel(e) \leftarrow;$
$Vowel(i) \leftarrow;$
$Vowel(o) \leftarrow;$
$Vowel(u) \leftarrow;$
$Vowel(y) \leftarrow;$

но **неравносильны** в операционной семантике : запрос

$$G :? Elem\_Vow(X, o \centerdot n \centerdot e \centerdot \mathbf{nil})$$

к $\mathcal{P}_1$ вычисляет $\theta_1 = \{X/o\}$ и $\theta_2 = \{X/e\}$,
а тот же запрос к $\mathcal{P}_2$ вычисляет только $\theta_1 = \{X/o\}$.

ОПЕРАТОР ОТСЕЧЕНИЯ

С введением в логические программы оператора отсечения теорема полноты операцинной семантики относительно декларативной семантики перестает быть справедливыми.

Поэтому оператором отсечения ! нужно пользоваться очень осторожно.

А что еще полезного и удобного
можно встроить в логические программы?

КОНЕЦ ЛЕКЦИИ 16.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 17.

Отрицание в логическом программировании. Оператор not. Встроенные предикаты и функции. Оператор вычисления значений. Модификация баз данных.

ОТРИЦАНИЕ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

Отрицание $\neg$ — это очень полезная логическая связка. Часто мы обращаемся с вопросами, используя отрицание.

Пример

$$
\begin{aligned}
\mathcal{P}: \quad & \text{птица}(\text{орел}) \leftarrow; \\
& \text{птица}(\text{воробей}) \leftarrow; \\
& \text{птица}(\text{пингвин}) \leftarrow; \\
& \text{летает}(\text{орел}) \leftarrow; \\
& \text{летает}(\text{воробей}) \leftarrow; \\
& \text{летает}(\text{самолет}) \leftarrow;
\end{aligned}
$$

Сформулируем вопрос: какая птица не летает?

$$G: ?\ \text{птица}(X), \neg\text{летает}(X)$$

Какой ответ мы ожидаем получить на этот запрос?

ОТРИЦАНИЕ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

Пример

С точки зрения декларативной семантики, для получения ответа на этот запрос нужно выяснить, выполняется ли логическое следование

$$\mathcal{P} \models \exists X(\text{птица}(X)\&\neg\text{летает}(X)),$$

$$\text{где } \mathcal{P} = \{\text{птица}(\text{орел}), \text{птица}(\text{воробей}), \text{птица}(\text{пингвин}), \text{летает}(\text{орел}), \text{летает}(\text{воробей}), \text{летает}(\text{самолет})\}$$

Оно не выполняется, т. к.

$$B_{\mathcal{P}} \models \mathcal{P},\ B_{\mathcal{P}} \not\models \exists X(\text{птица}(X)\&\neg\text{летает}(X)).$$

Значит, ответ на этот запрос должен быть отрицательным: такой птицы нет .

ОТРИЦАНИЕ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

Пример

Однако в обыденной жизни мы в таких случаях даем совсем другой ответ.

Обнаружив, что в нашей базе знаний $\mathcal{P}$
есть сведение $\text{птица}(\text{пингвин})$ и
нет никаких сведений о том, что $\text{летает}(\text{пингвин})$,
мы отвечаем:

«Насколько позволяет судить наша база знаний, нелетающая птица — это пингвин».

В юрипруденции такой подход к решению вопроса о виновности человек называется **презумпцией невиновности** :

в случае отсутствия свидетельств виновности
человек считается невиновным.

ОТРИЦАНИЕ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

Мы можем распространить принцип презумпции невиновности и на логические программы.

Допущение Замкнутости Мира

Пусть имеется некоторое непротиворечивое множество замкнутых формул Γ (например, хорновская логическая программа) и замкнутая формула φ (например, запрос или отдельная подцель).

Тогда формула $\neg\varphi$ является логическим следствием множества Γ в допущении замкнутости мира

$$\Gamma \models_{CWA} \neg\varphi,$$

если **неверно**, что φ логически следует из Γ, т. е. $\Gamma \not\models \varphi$.

Здесь CWA — аббревиатура Closed World Assumption.

ОТРИЦАНИЕ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

Суть Допущения Замкнутости Мира (CWA) состоит в том, что при извлечении CWA-логических следствий из базы знаний Γ ($\models_{CWA}$) нужно рассматривать не все модели для Γ, а только такую минимальную модель, в которой истинными являются одни лишь классические следствия ($\models$) из Γ.

Такая минимальная модель существует, вообще говоря, не всегда (например, если $\Gamma = \{A \vee B\}$).

Но в случае хорновских логических программ, минимальной моделью для программы $\mathcal{P}$ является наименьшая эрбрановская модель $\mathbf{M}_{\mathcal{P}}$.

ОТРИЦАНИЕ В ЛОГИЧЕСКОМ ПРОГРАММИРОВАНИИ

Обратимся к орнитологическому примеру.

Пример

$$\begin{aligned}\mathcal{P} &= \{\text{птица}(\text{орел}), \text{птица}(\text{воробей}), \text{птица}(\text{пингвин}),\\ &\quad \text{летает}(\text{орел}), \text{летает}(\text{воробей}), \text{летает}(\text{самолет})\}\end{aligned}$$

$$\varphi = \text{птица}(\text{пингвин}) \;\&\; \neg\text{летает}(\text{пингвин})$$

Тогда

$\mathcal{P} \models_{CWA} \text{птица}(\text{пингвин})$, поскольку $\mathbf{M}_{\mathcal{P}} \models \text{птица}(\text{пингвин})$,

$\mathcal{P} \models_{CWA} \neg\text{летает}(\text{пингвин})$, поскольку $\mathbf{M}_{\mathcal{P}} \not\models \text{летает}(\text{пингвин})$.

Значит, $\mathcal{P} \models_{CWA} \text{птица}(\text{пингвин}) \;\&\; \neg\text{летает}(\text{пингвин})$.

ОПЕРАТОР not

Итак, теперь к логическим программам можно обращаться с запросами, содержащими отрицания подцелей, и декларативная семантика умеет разумно обращаться с таким отрицательными подцелями.

Нам остается научить этому обращению компьютер, т. е. ввести в операционную семантику правила для обработки отрицательных подцелей.

Однако здесь нас ожидает неприятный сюрприз.

ОПЕРАТОР not

Предположим, что имеется запрос $G : ?\neg C_0$ к программе $\mathcal{P}$.

Чтобы получить утвердительный ответ на запрос G (т. е. доказать, что $\mathcal{P} \models_{CWA} \neg C_0$), интерпретатор должен проверить, что $\mathbf{M}_{\mathcal{P}} \not\models C_0$.

Для этого интерпретатор должен убедиться, что запрос $G' : ?C_0$ к программе $\mathcal{P}$ **не имеет ни одного успешного вычисления**.

Но эта задача является алгоритмически неразрешимой (почему?).
Да потому, что к ней сводится проблема останова машин Тьюринга.

Следовательно, никакой интерпретатор логических программ не может гарантировать получение утвердительного ответа на запрос $G : ?\neg C_0$ даже в том случае, если этот ответ существует.

ОПЕРАТОР not

Алгоритмическая неразрешимость проблемы существования (отсутствия) успешного вычисления логических программ — это непреодолимое препятствие на пути создания такой операционной семантики, которая была бы адекватна декларативной семантике в рамках Допущения Замкнутости Мира.

Можно лишь построить такой интерпретатор (операционной семантики), который как можно лучше соответствует CWA при обращении с отрицательными подцелями $\neg C_0$.

Для этого в язык логического программирования был введен специальный (встроенный) оператор **not**.

ОПЕРАТОР not

Аргументами оператора **not** являются атомы.

Для вычисления ответов на запрос ? **not**(C_0) вводится правило SLDNF-резолюции (Not as Failure), которое определяется следующим образом.

Правило SLDNF-резолюции

Пусть имеется запрос G_0 : ? **not**$(C_0), C_1, \ldots, C_n$ к программе $\mathcal{P}$.

Для вычисления SLDNF-резольвенты G_1

1. формируется запрос G' : ? C_0 к программе $\mathcal{P}$;
2. проводится построение (обход) дерева вычислений T запроса G' : ? C_0;
3. в зависимости от устройства дерева T возможен один из трех исходов.

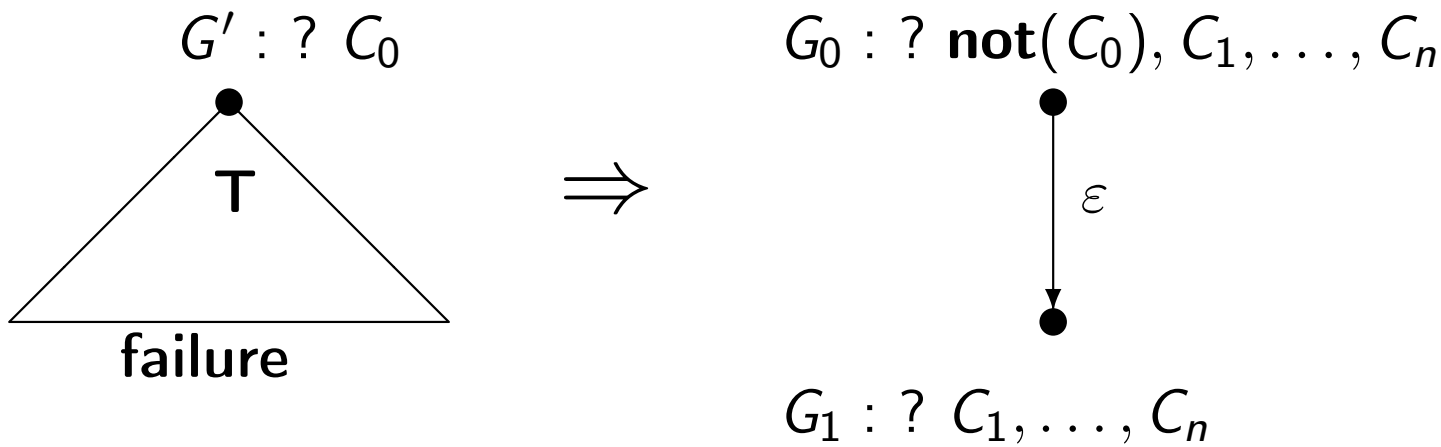
ОПЕРАТОР not

Правило SLDNF-резолюции

Вариант 1: **Успех.**

Дерево T конечно, и все его ветви (SLD-резолютивные вычисления) являются тупиковыми.

Тогда запрос $G_0 : ?\ \mathbf{not}(C_0), C_1, \ldots, C_n$ имеет SLDNF-резольвенту $G_1 = ?\ C_1, \ldots, C_n$, которая получается с использованием пустой подстановки ε.



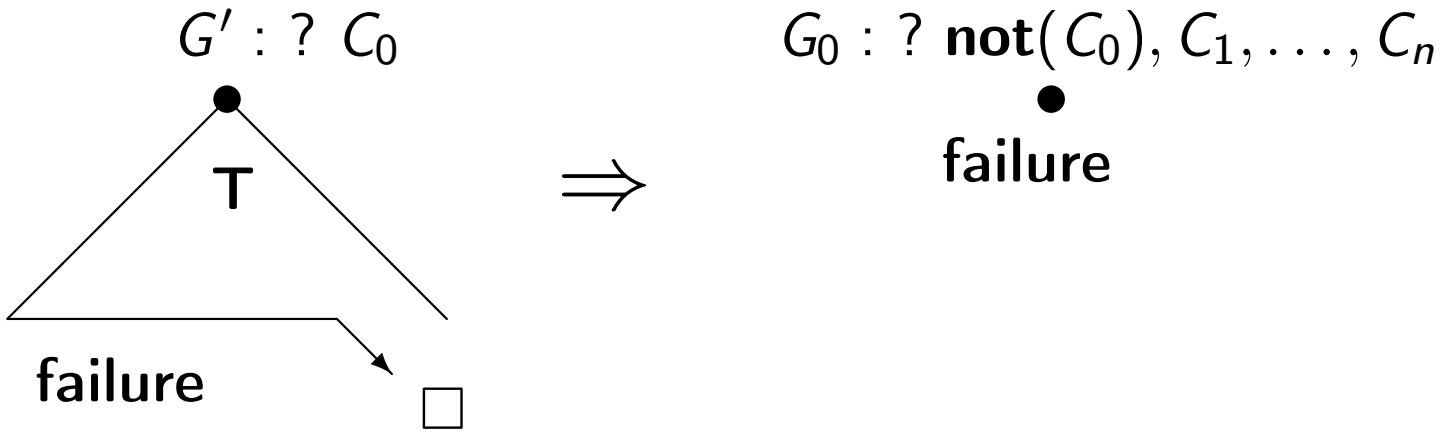
ОПЕРАТОР not

Операционная семантика оператор not

Вариант 2: **Неудача.**

При построении (обходе) дерева T было обнаружено успешное вычисление.

Тогда запрос $G_0 : ?\ \mathbf{not}(C_0), C_1, \ldots, C_n$ не имеет SLDNF-резольвент, и вычисление этого запроса является тупиковым .



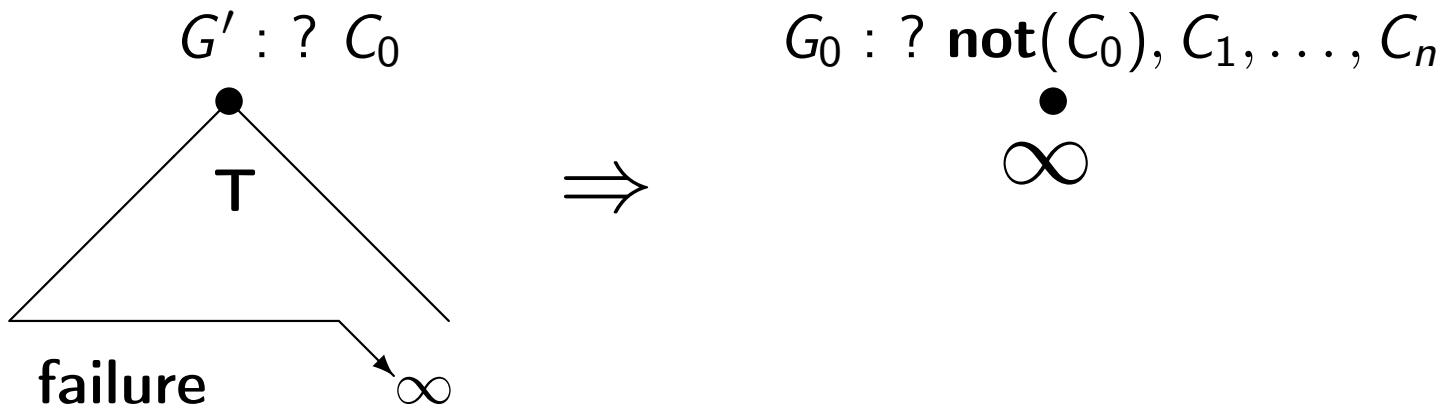
ОПЕРАТОР not

Операционная семантика оператор not

Вариант 3: **Бесконечное вычисление.**

Дерево T бесконечно и при его построении (обходе) не было обнаружено успешных вычислений.

Тогда запрос $G_0 : ?\ \mathbf{not}(C_0), C_1, \ldots, C_n$ не имеет SLDNF-резольвент, и вычисление этого запроса является бесконечным («сингулярная бесконечность»).



ОПЕРАТОР not

Операционная семантика оператор not

Вариант 3: **Бесконечное вычисление.**

Дерево T бесконечно и при его построении (обходе) не было обнаружено успешных вычислений.

Тогда запрос $G_0 : ?\ \mathbf{not}(C_0), C_1, \ldots, C_n$ не имеет SLDNF-резольвент, и вычисление этого запроса является бесконечным («сингулярная бесконечность»).

Условие существования бесконечного вычисления запроса $\mathbf{not}(C_0)$ описано не вполне строго, поскольку обнаружение успешного вычисления существенно зависит от стратегии обхода дерева SLD-резолютивных вычислений. Так, например, стандартная стратегия обхода в глубину может не обнаружить существующего успешного вычисления (почему?).

ОПЕРАТОР not

Теорема (корректности SLDNF-резолюции)

Если запрос G : ? **not**(C_0) к хорновской логической программе $\mathcal{P}$ имеет успешное SLDNF-резолютивное вычисление, то $\mathcal{P} \models_{CWA} \neg\exists\overline{y}\ C_0$.

Доказательство

Самостоятельно .

А вот обратное утверждение (теорема полноты) для SLDNF-резолютивного вычисления будет уже неверно.

ОПЕРАТОР not

Пример.

Рассмотрим программу поиска всех тех букв X слова L', которые не содержатся в слове L''.

$$\begin{array}{lrcl} \mathcal{P}: & S(X, L', L'') & \leftarrow & E(X, L'), \mathbf{not}(E(X, L'')); \\ & E(X, X \,.\, L) & \leftarrow & ; \\ & E(X, Y \,.\, L) & \leftarrow & E(X, L); \end{array}$$

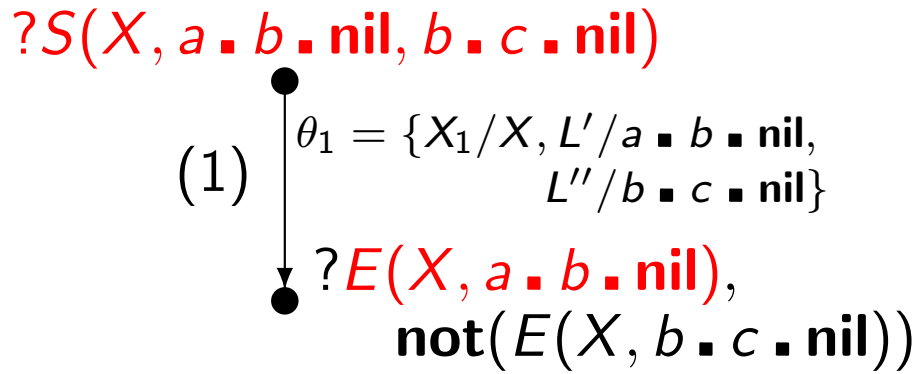
и обратимся к ней с запросом

$$G_0 : ?\ S(X, a \,.\, b \,.\, \mathbf{nil}, b \,.\, c \,.\, \mathbf{nil}).$$

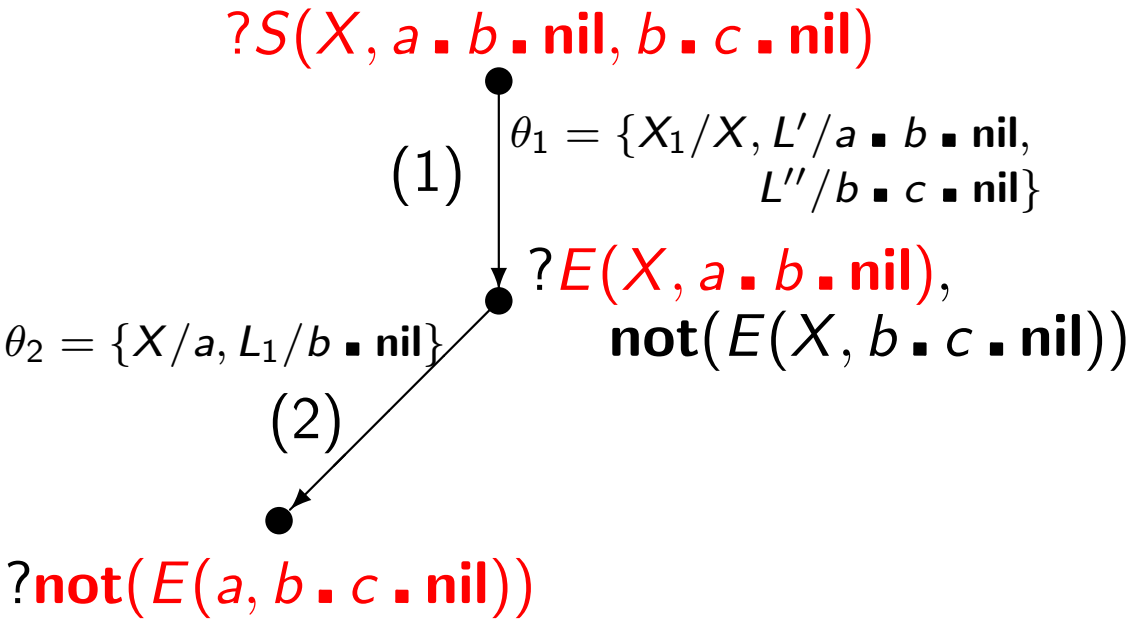
$$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L'')); \quad (1)$$
$$E(X, X \centerdot L) \leftarrow; \quad (2)$$
$$E(X, Y \centerdot L) \leftarrow E(X, L); \quad (3)$$

$$?S(X, a \centerdot b \centerdot \mathbf{nil}, b \centerdot c \centerdot \mathbf{nil})$$

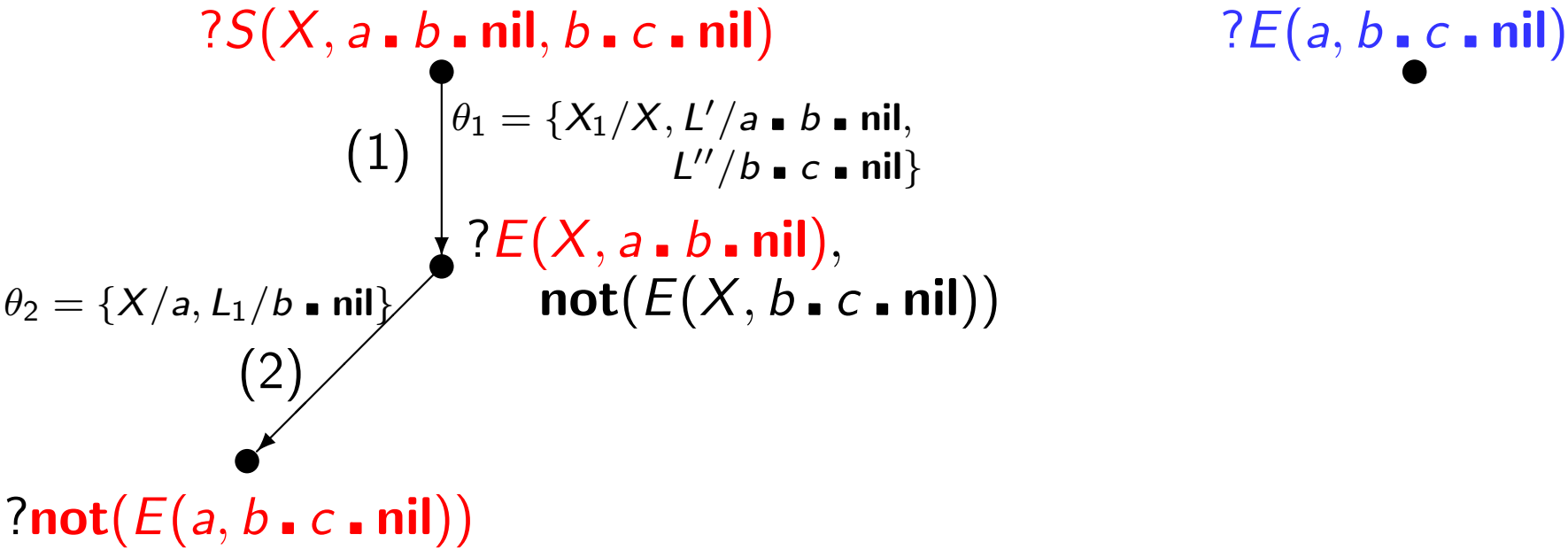
$$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L'')); \quad (1)$$
$$E(X, X \centerdot L) \leftarrow; \quad (2)$$
$$E(X, Y \centerdot L) \leftarrow E(X, L); \quad (3)$$



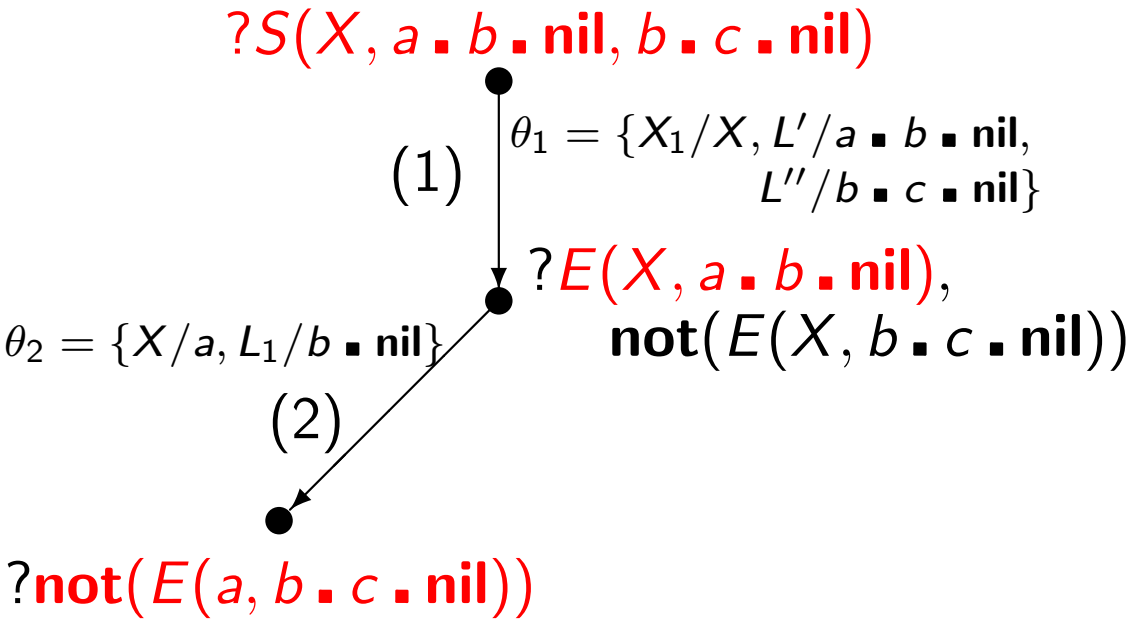
$$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L'')); \quad (1)$$
$$E(X, X \centerdot L) \leftarrow; \quad (2)$$
$$E(X, Y \centerdot L) \leftarrow E(X, L); \quad (3)$$

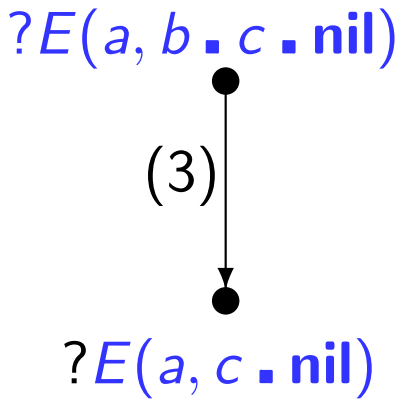


$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \centerdot L) \leftarrow$; (2)
$E(X, Y \centerdot L) \leftarrow E(X, L)$; (3)

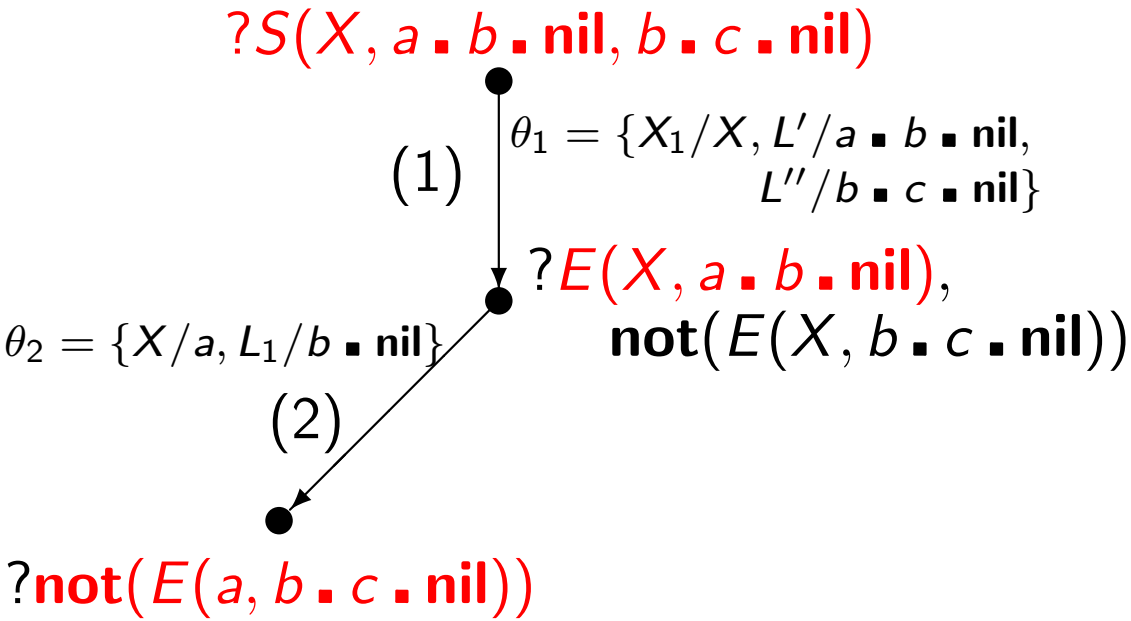


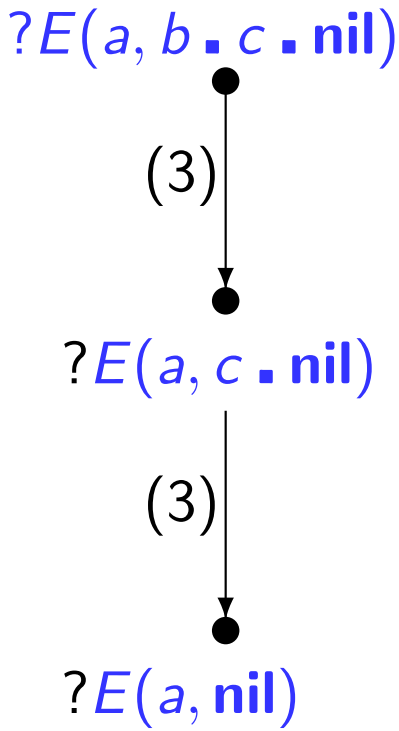
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



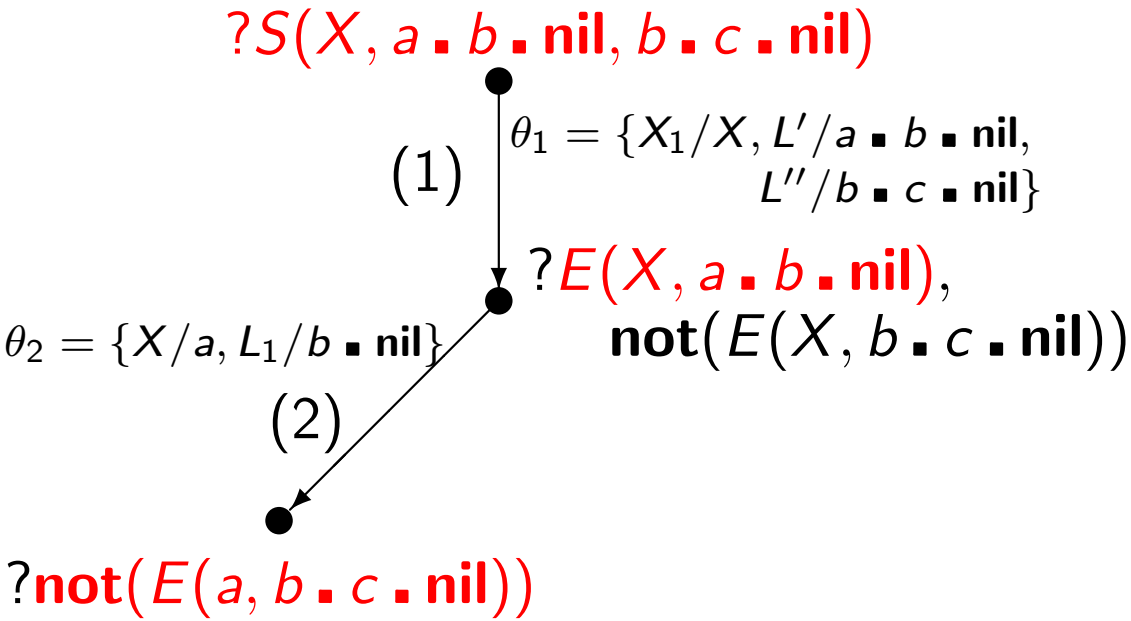


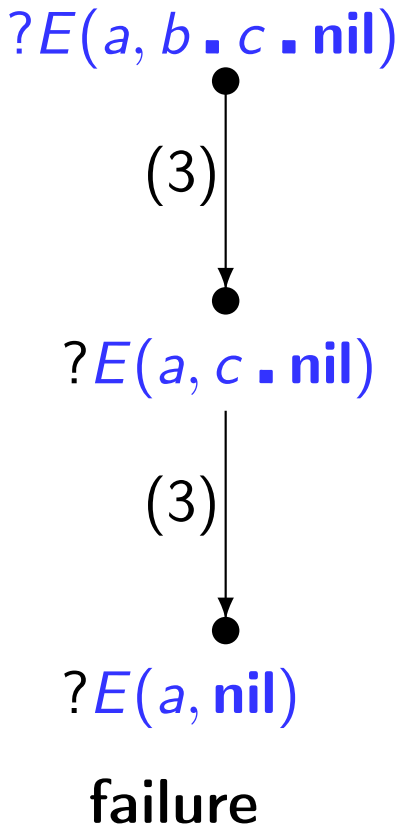
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



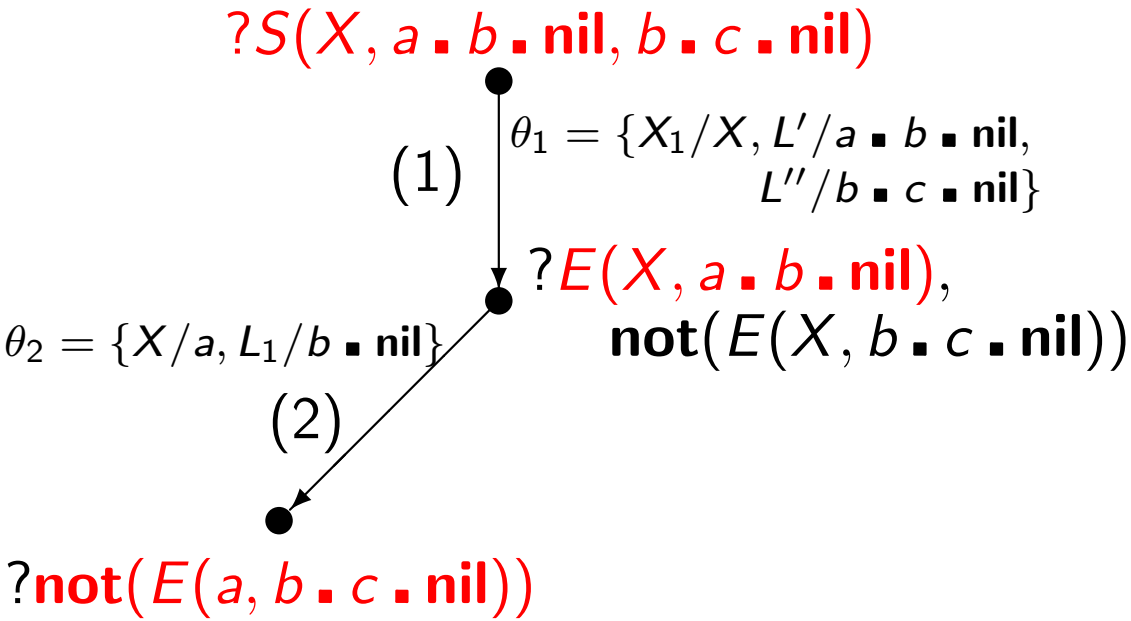


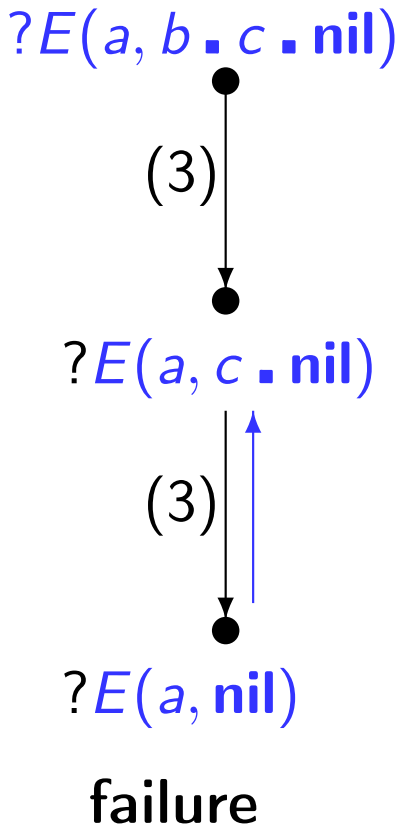
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



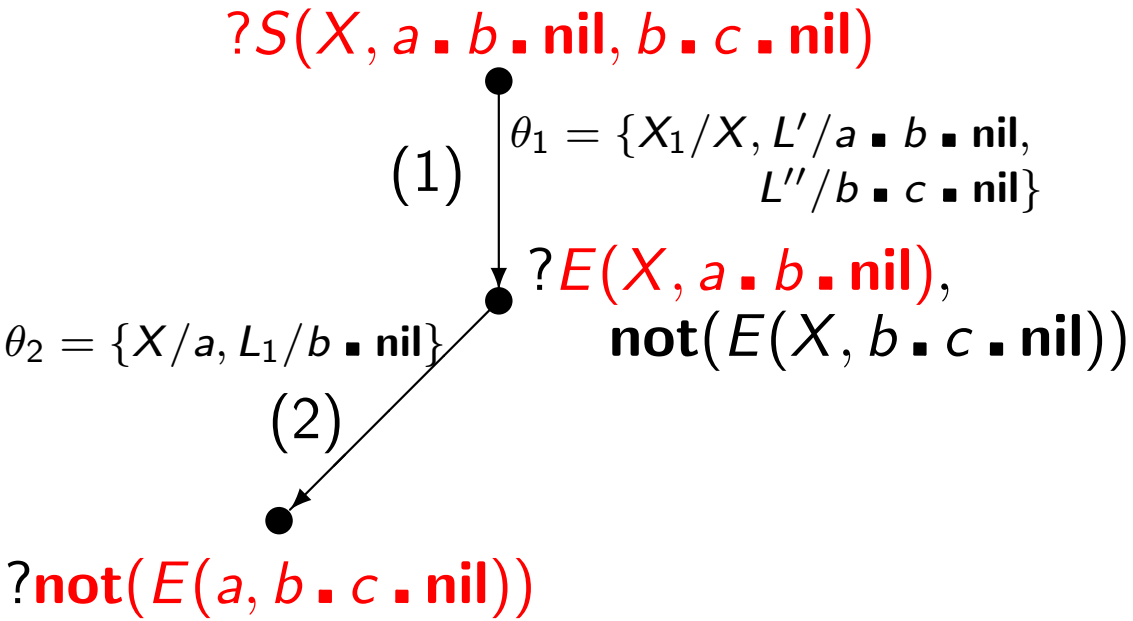


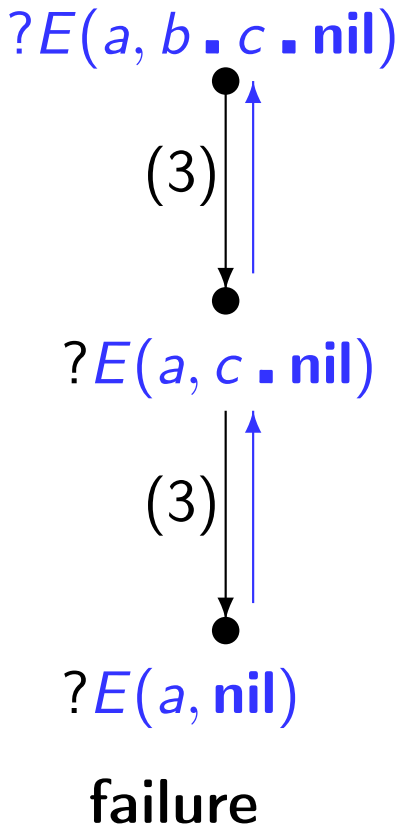
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



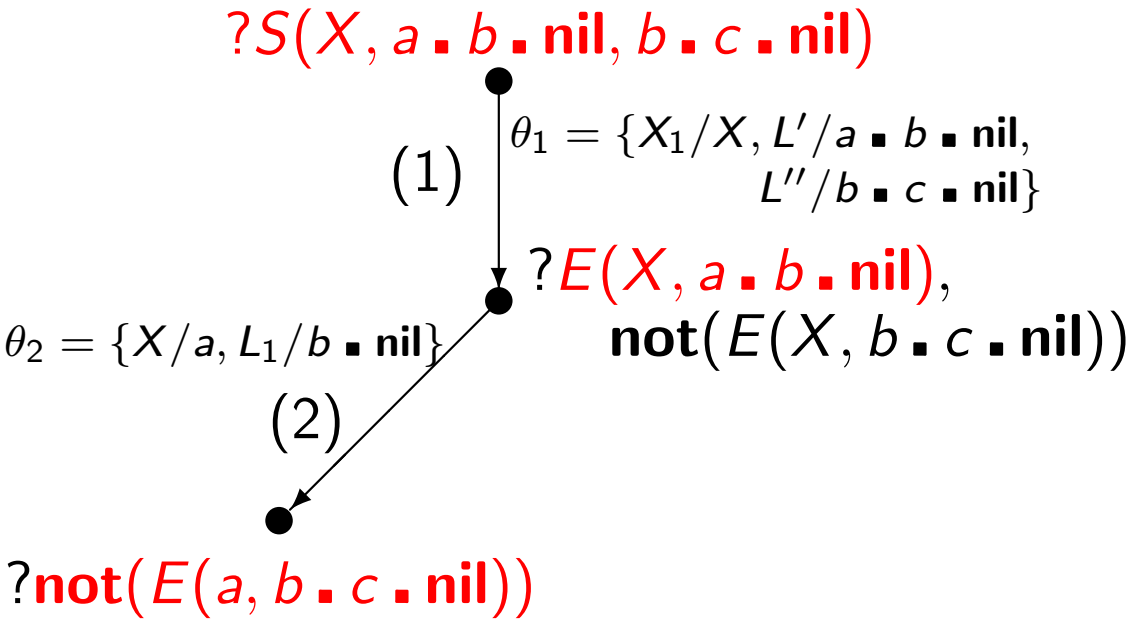


$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)

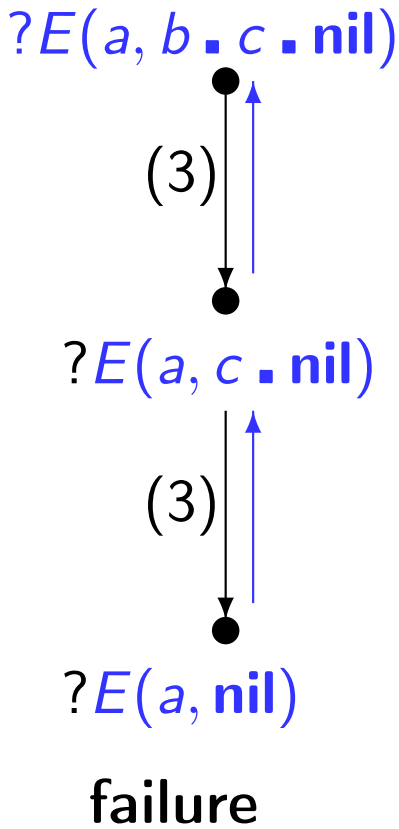




$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \blacksquare L) \leftarrow;$ (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L);$ (3)

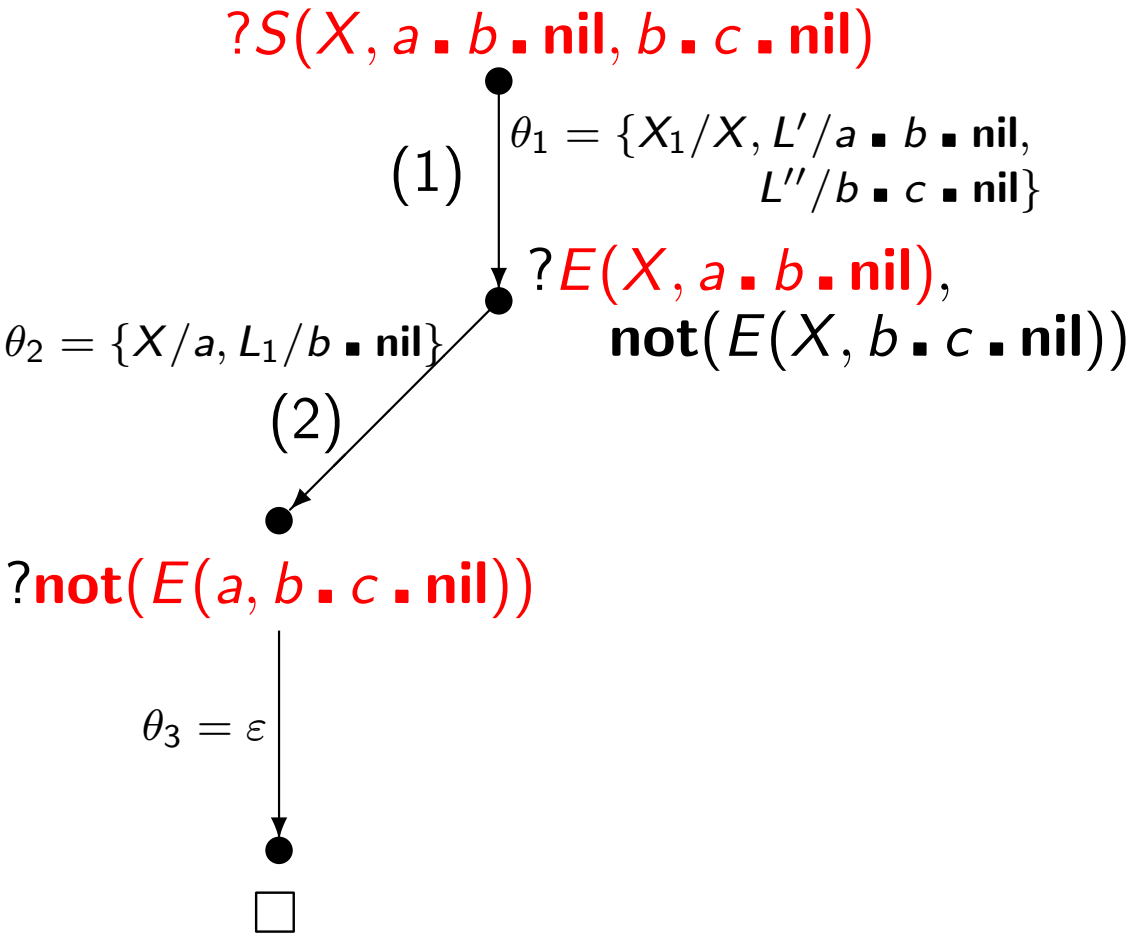


Успех



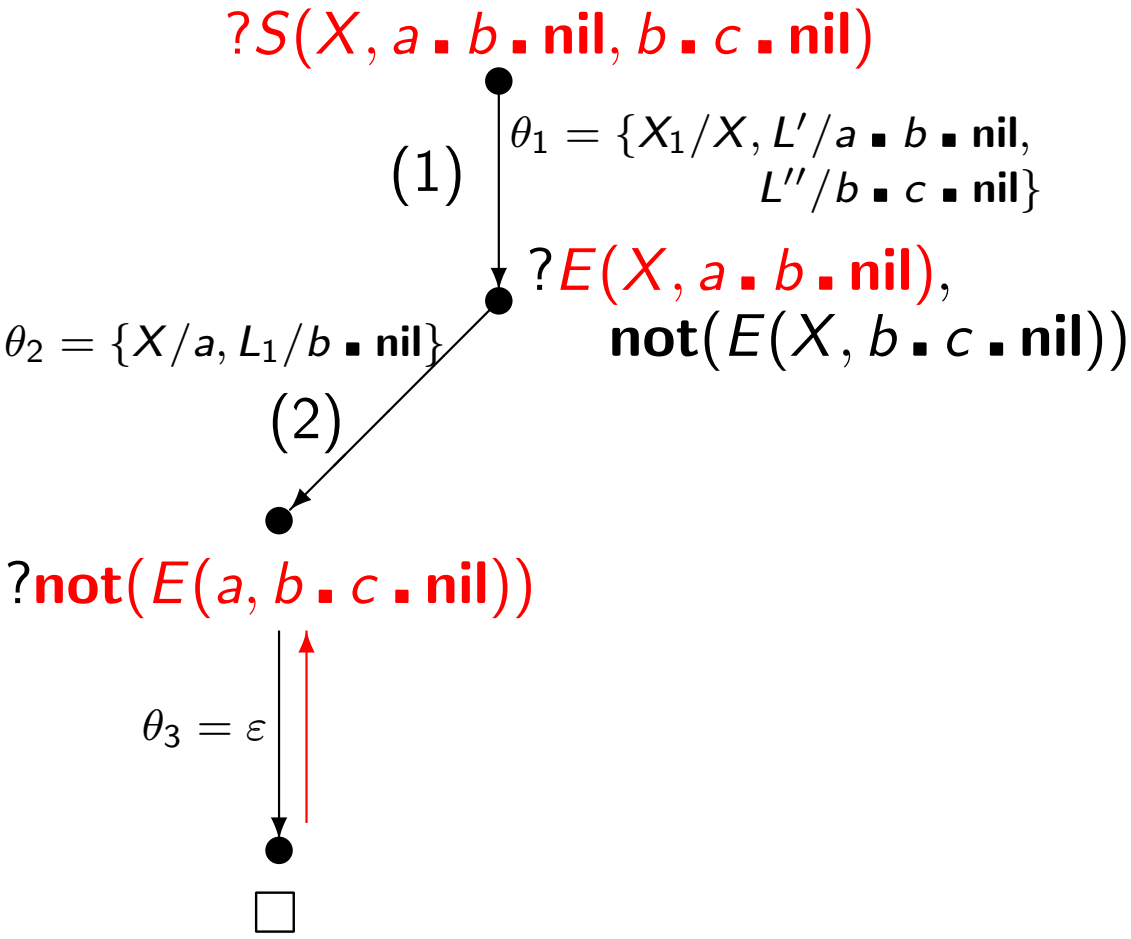
failure

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \centerdot L) \leftarrow$; (2)
$E(X, Y \centerdot L) \leftarrow E(X, L)$; (3)



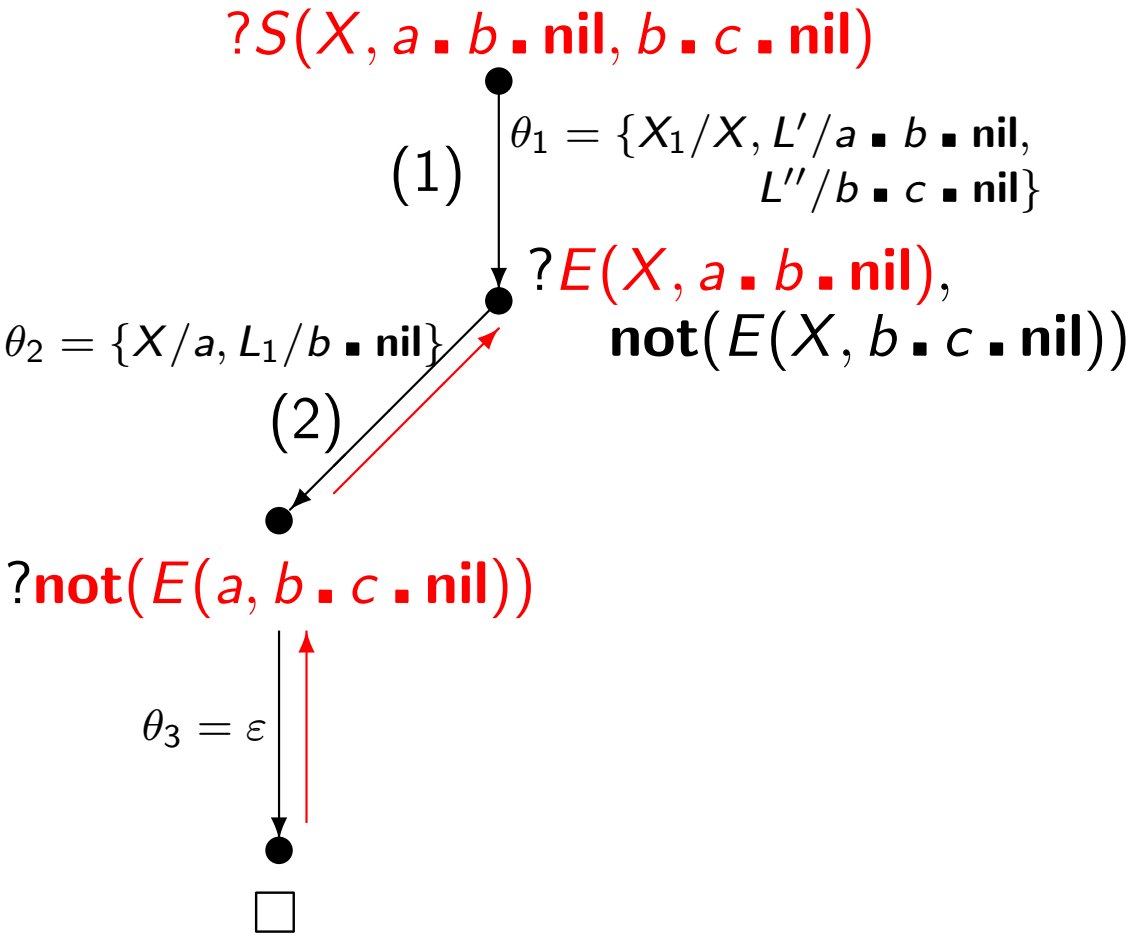
Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \blacksquare L) \leftarrow$; (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L)$; (3)



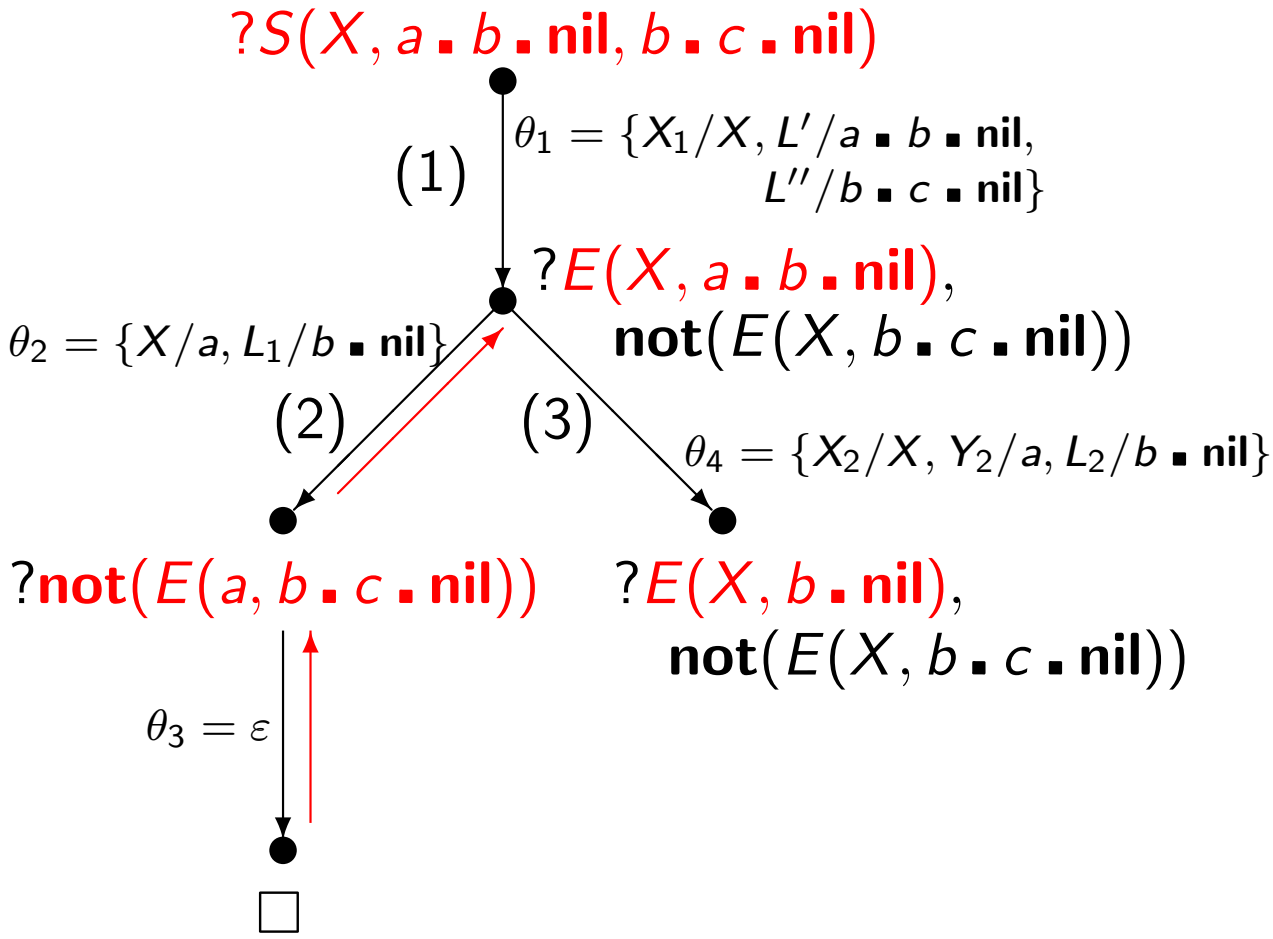
Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \blacksquare L) \leftarrow$; (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L)$; (3)



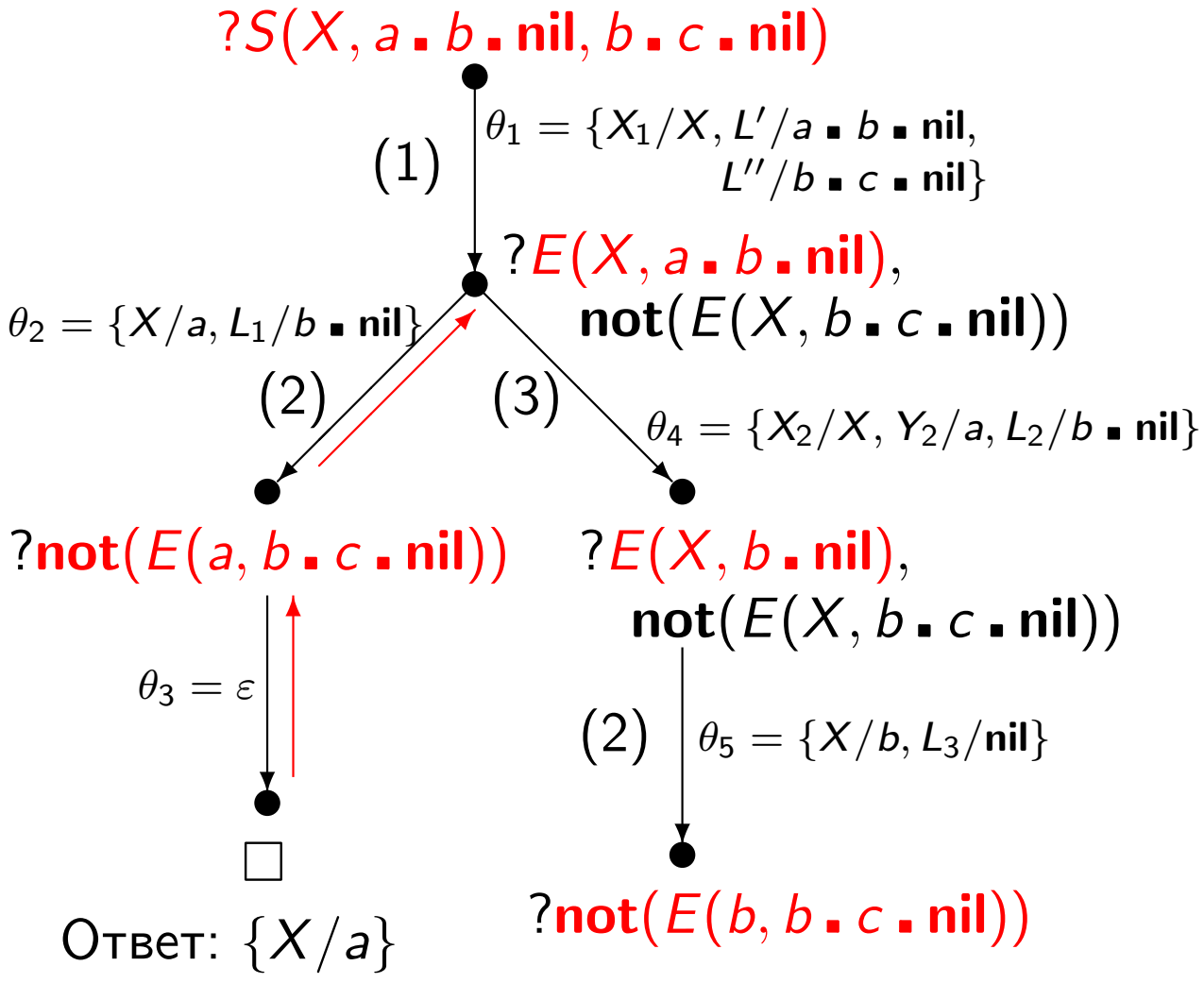
Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \blacksquare L) \leftarrow$; (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L)$; (3)



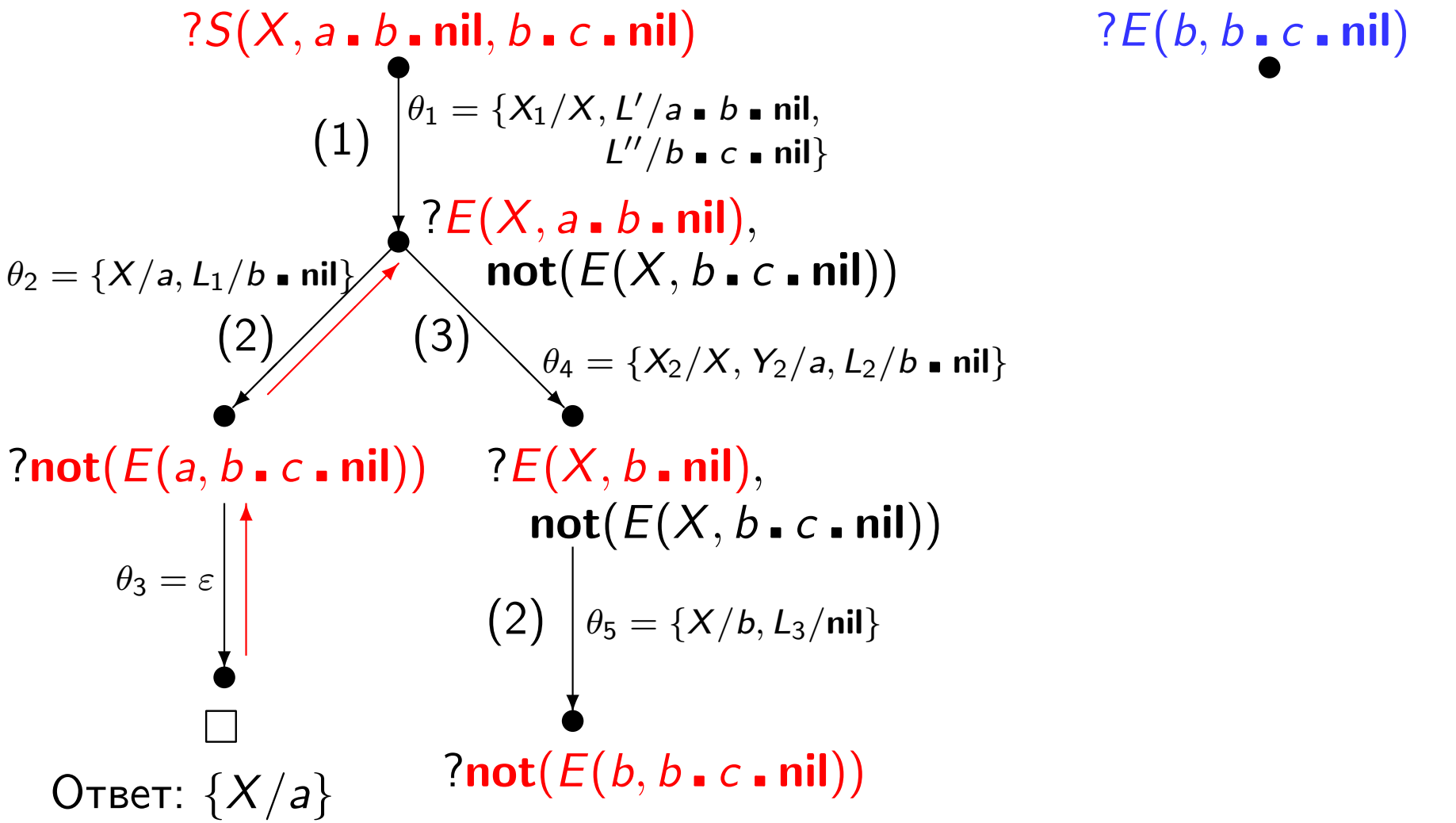
Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \blacksquare L) \leftarrow$; (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L)$; (3)

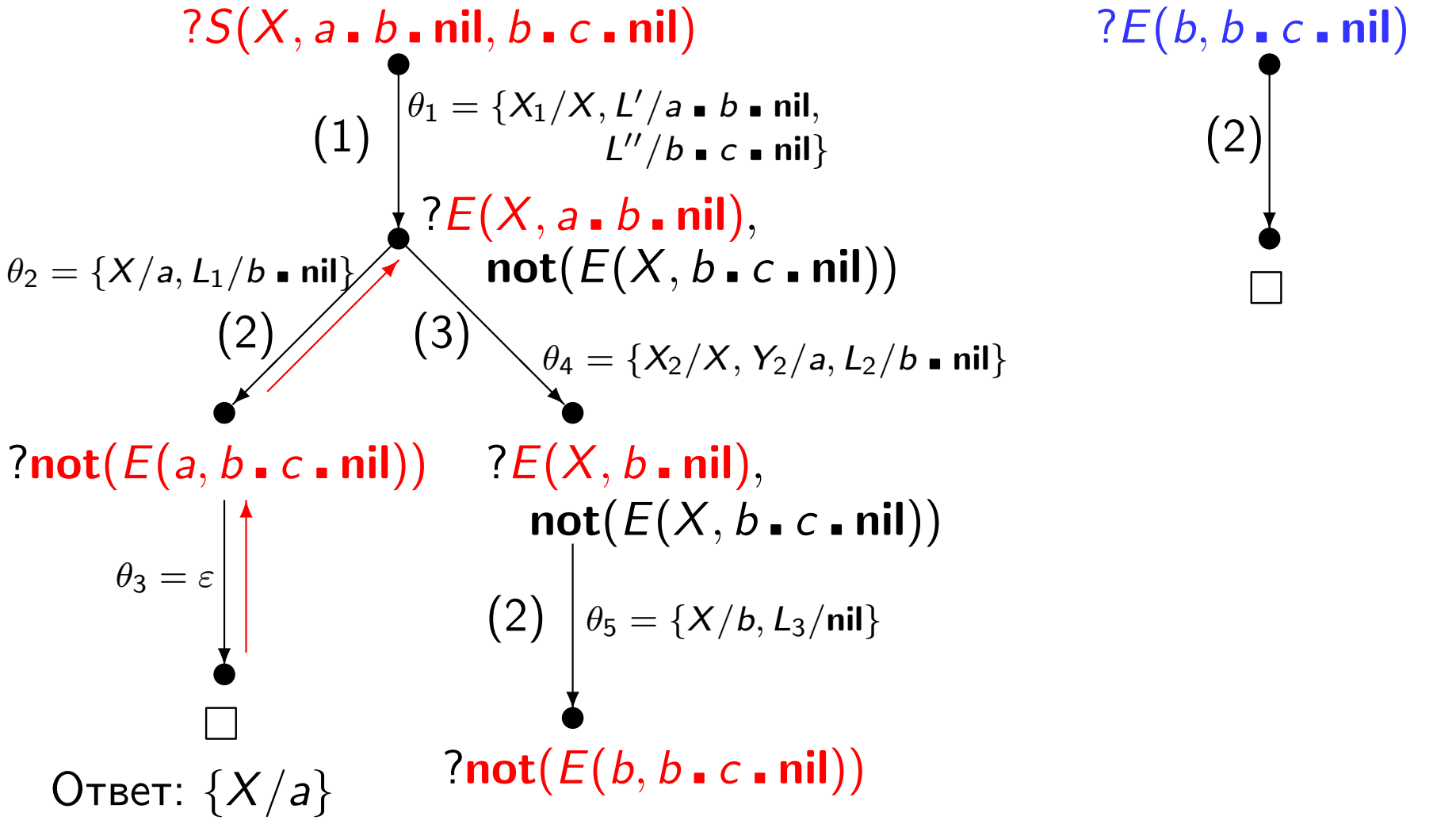


Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \centerdot L) \leftarrow$; (2)
$E(X, Y \centerdot L) \leftarrow E(X, L)$; (3)



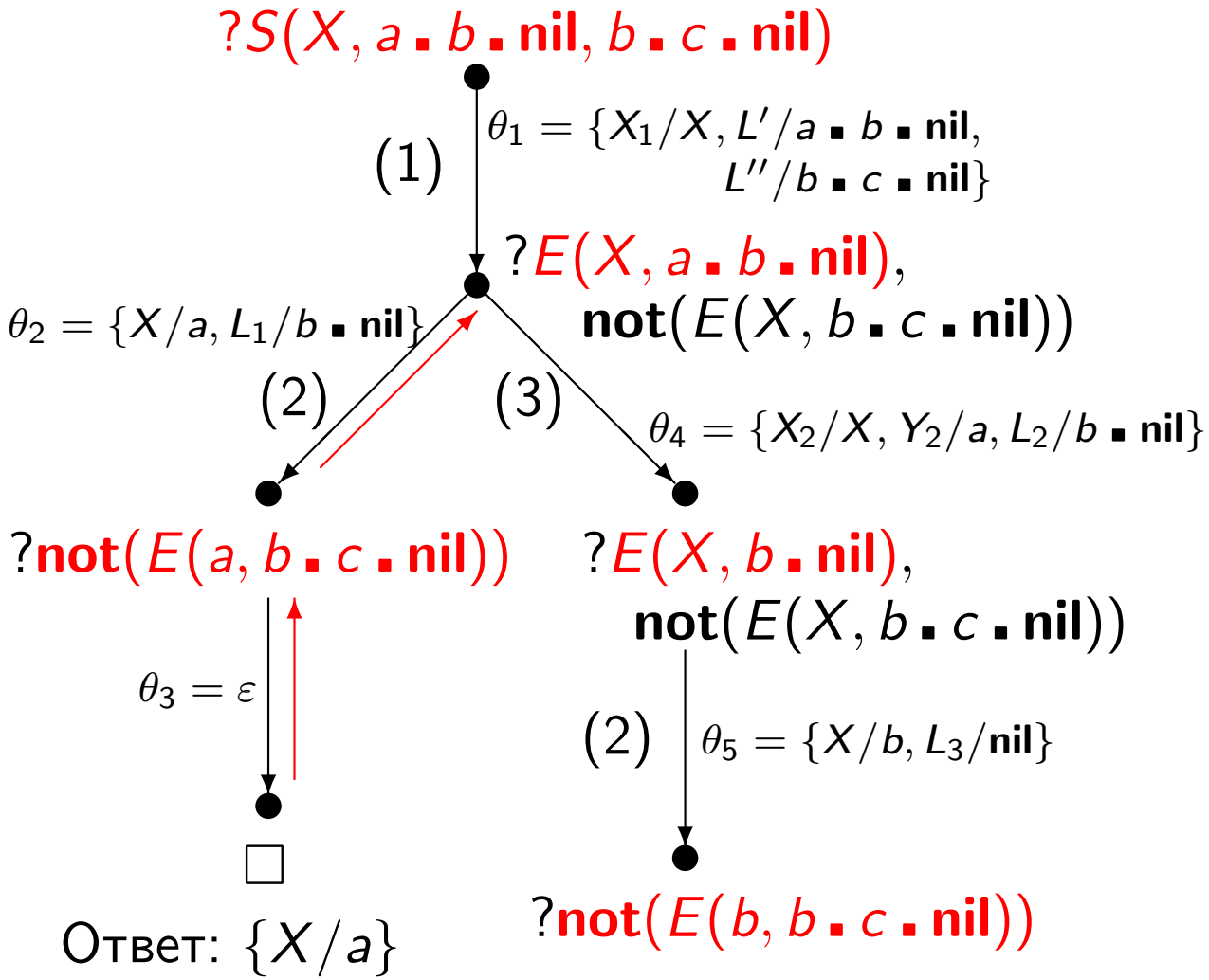
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \centerdot L) \leftarrow$; (2)
$E(X, Y \centerdot L) \leftarrow E(X, L)$; (3)



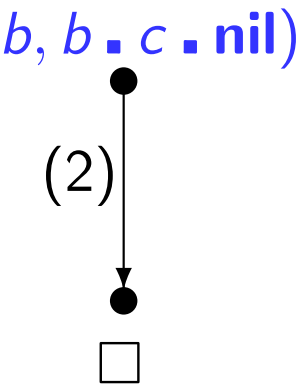
Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \blacksquare L) \leftarrow;$ (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L);$ (3)

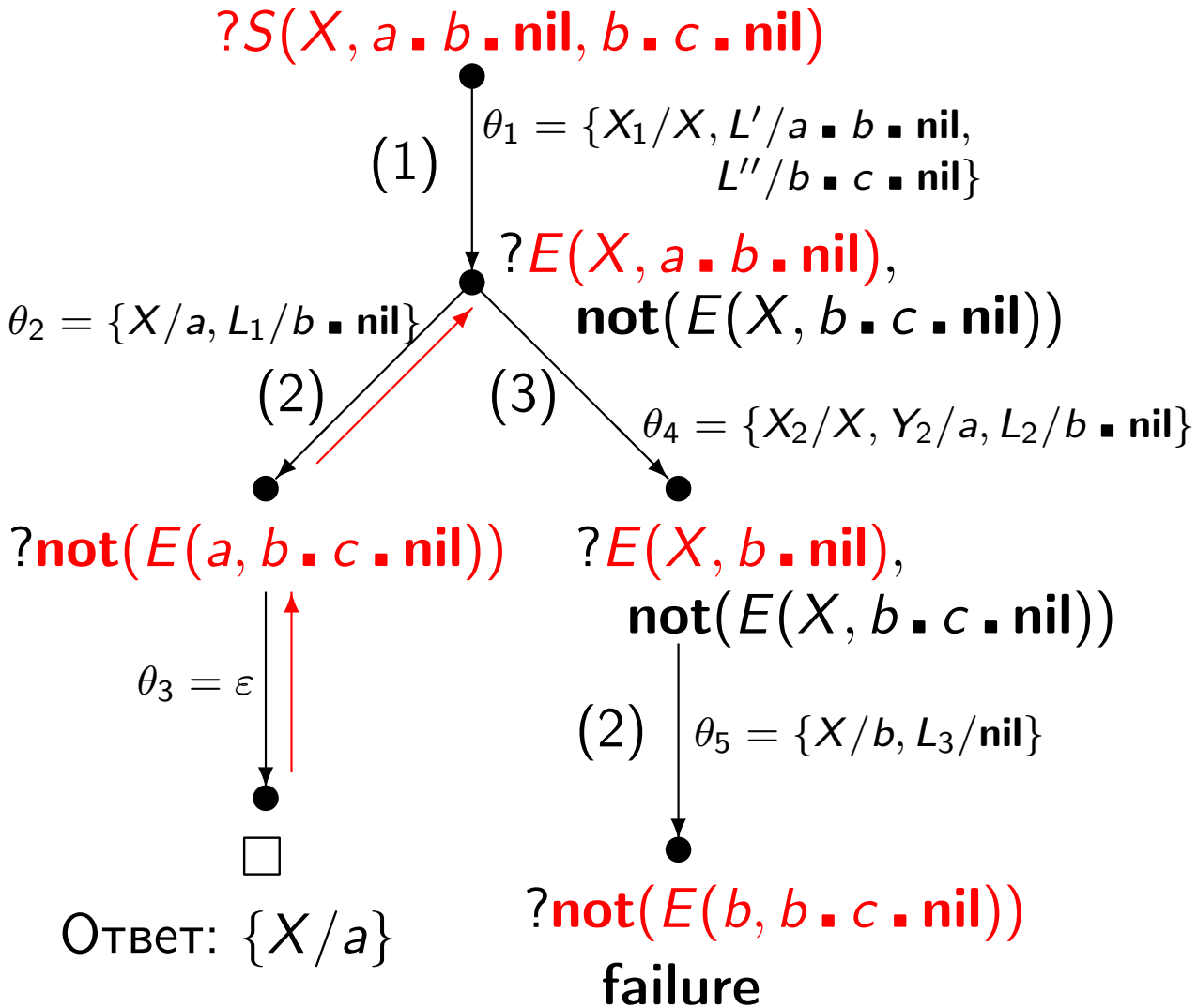
Неудача



Ответ: $\{X/a\}$

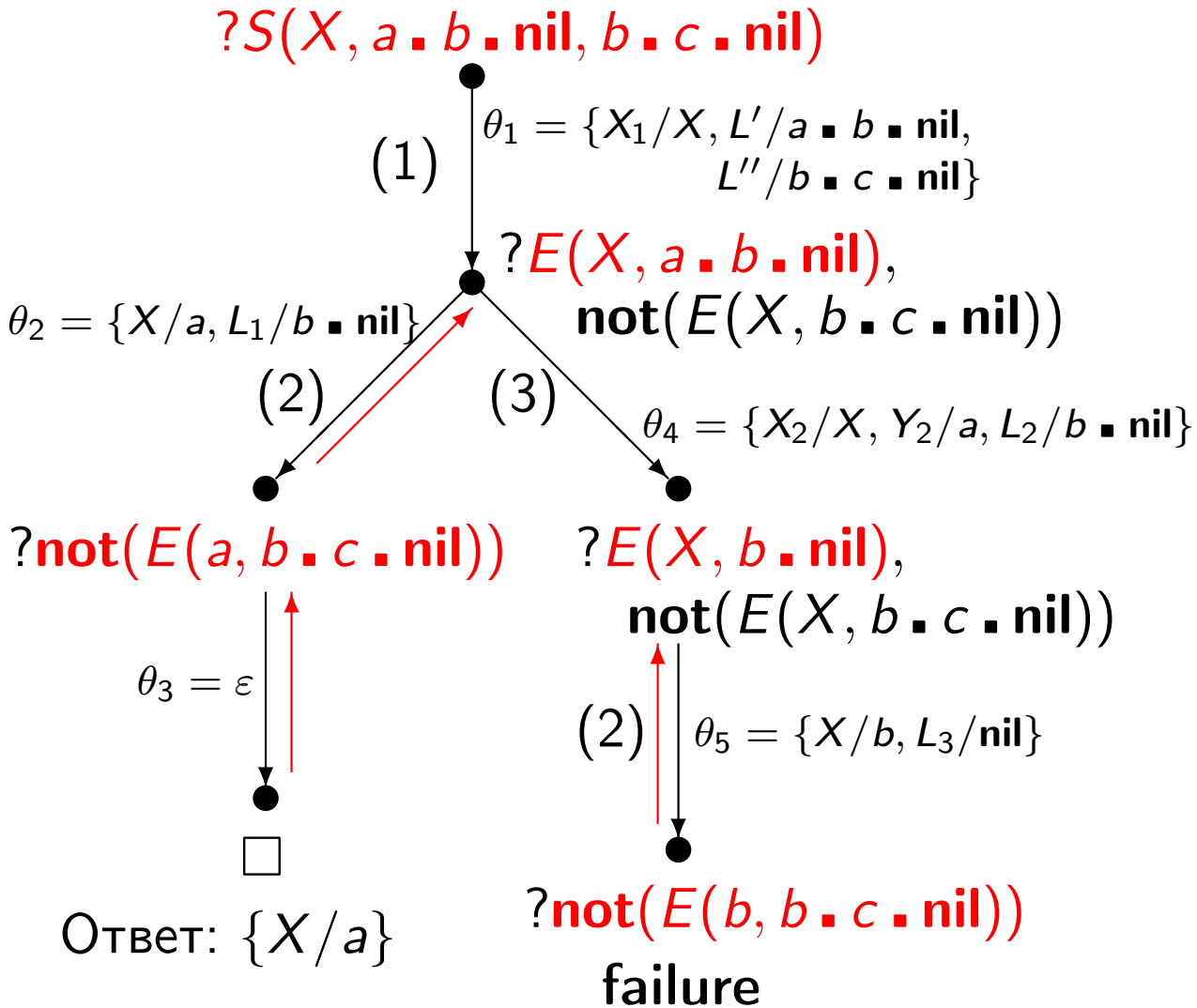


$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \blacksquare L) \leftarrow$; (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L)$; (3)



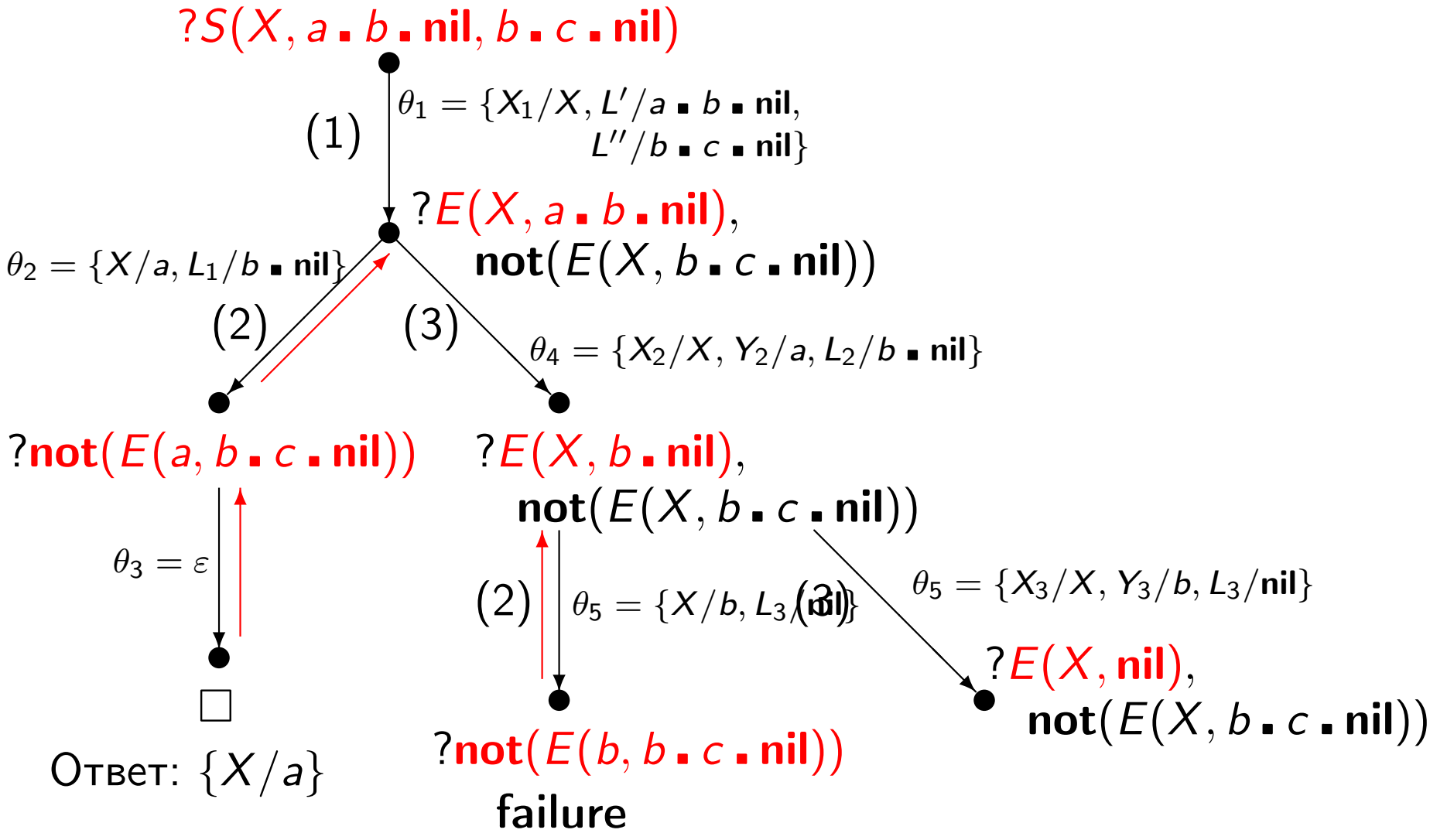
Ответ: $\{X/a\}$

$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \blacksquare L) \leftarrow$; (2)
$E(X, Y \blacksquare L) \leftarrow E(X, L)$; (3)

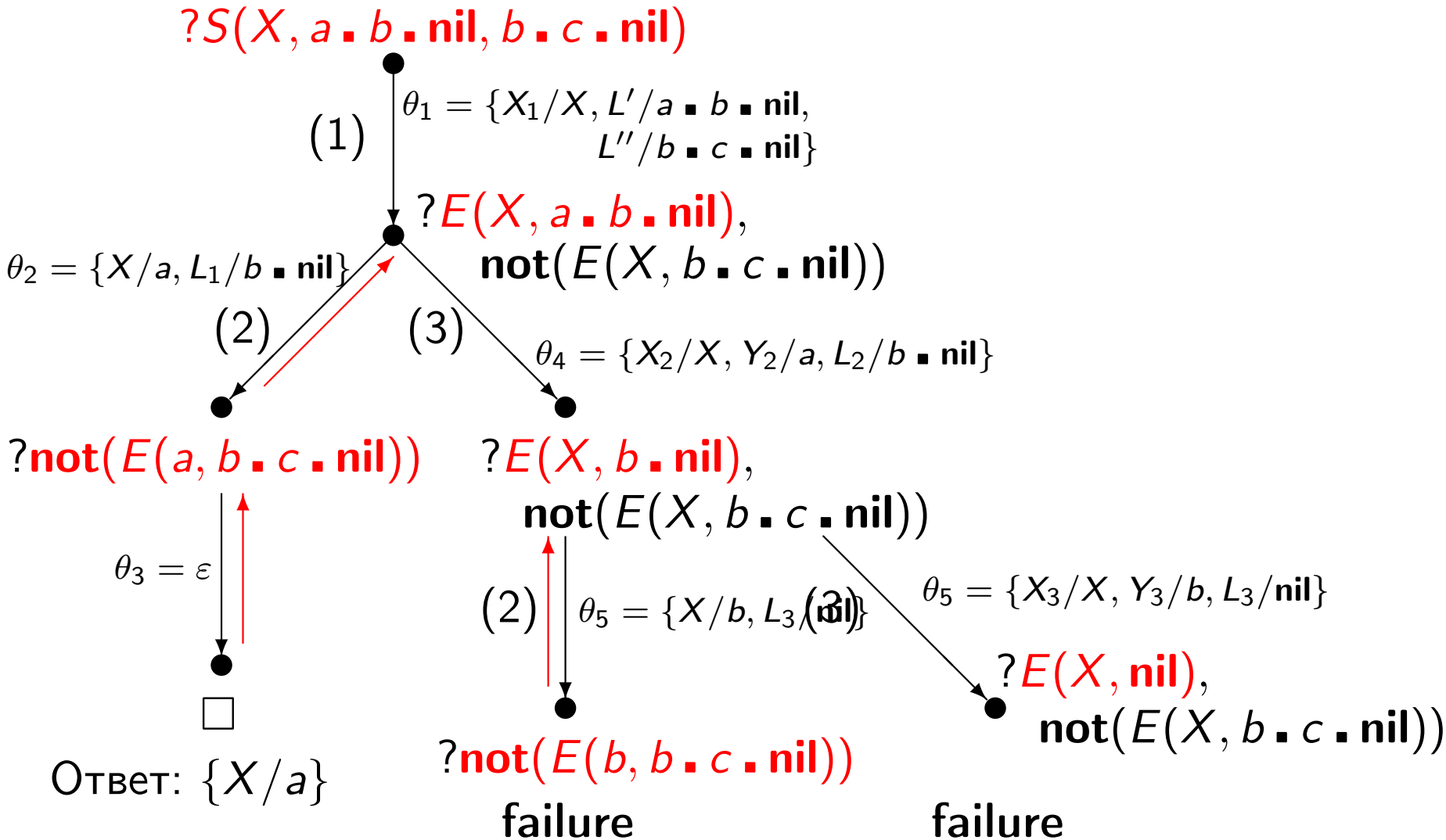


Ответ: $\{X/a\}$

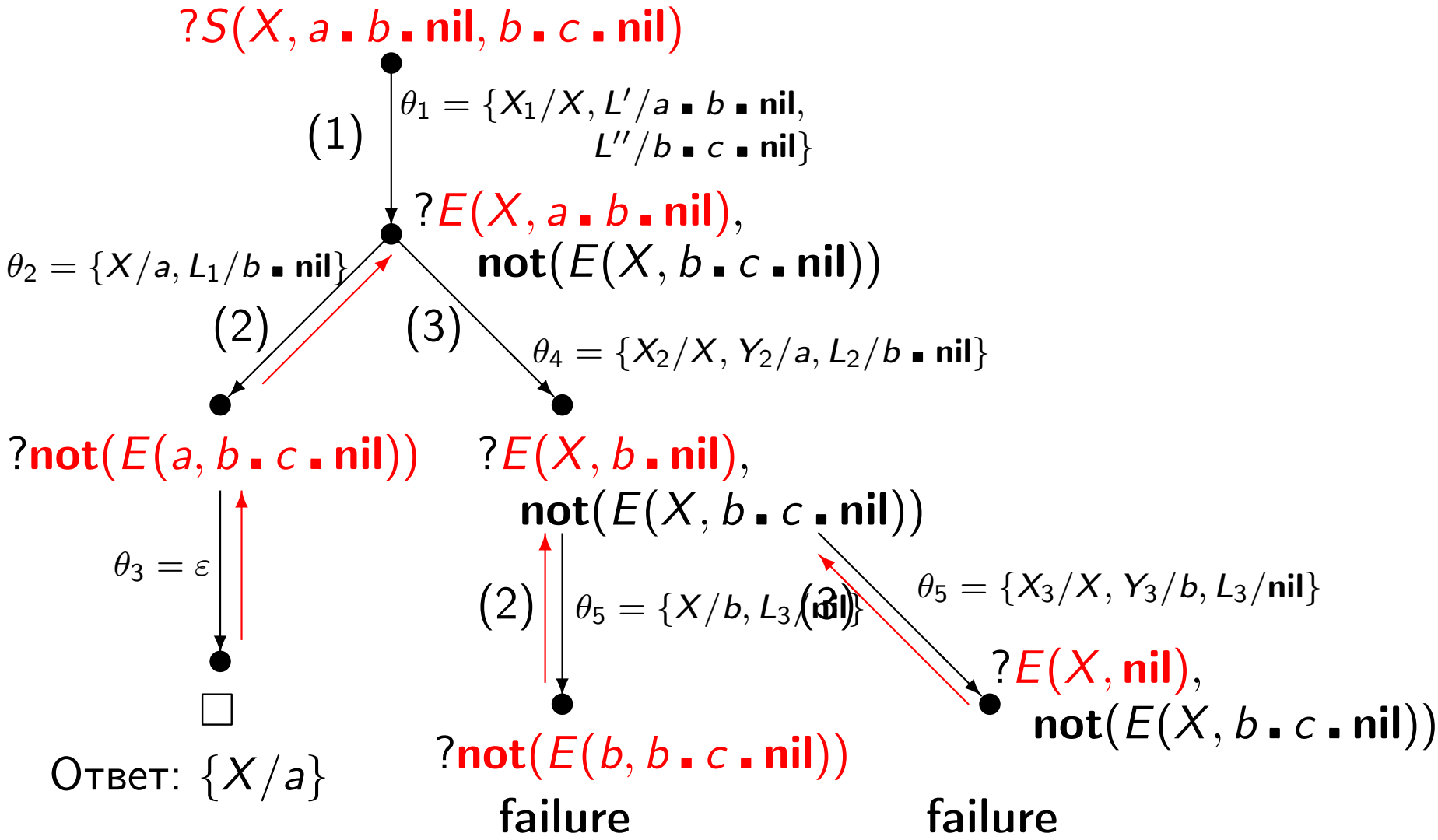
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



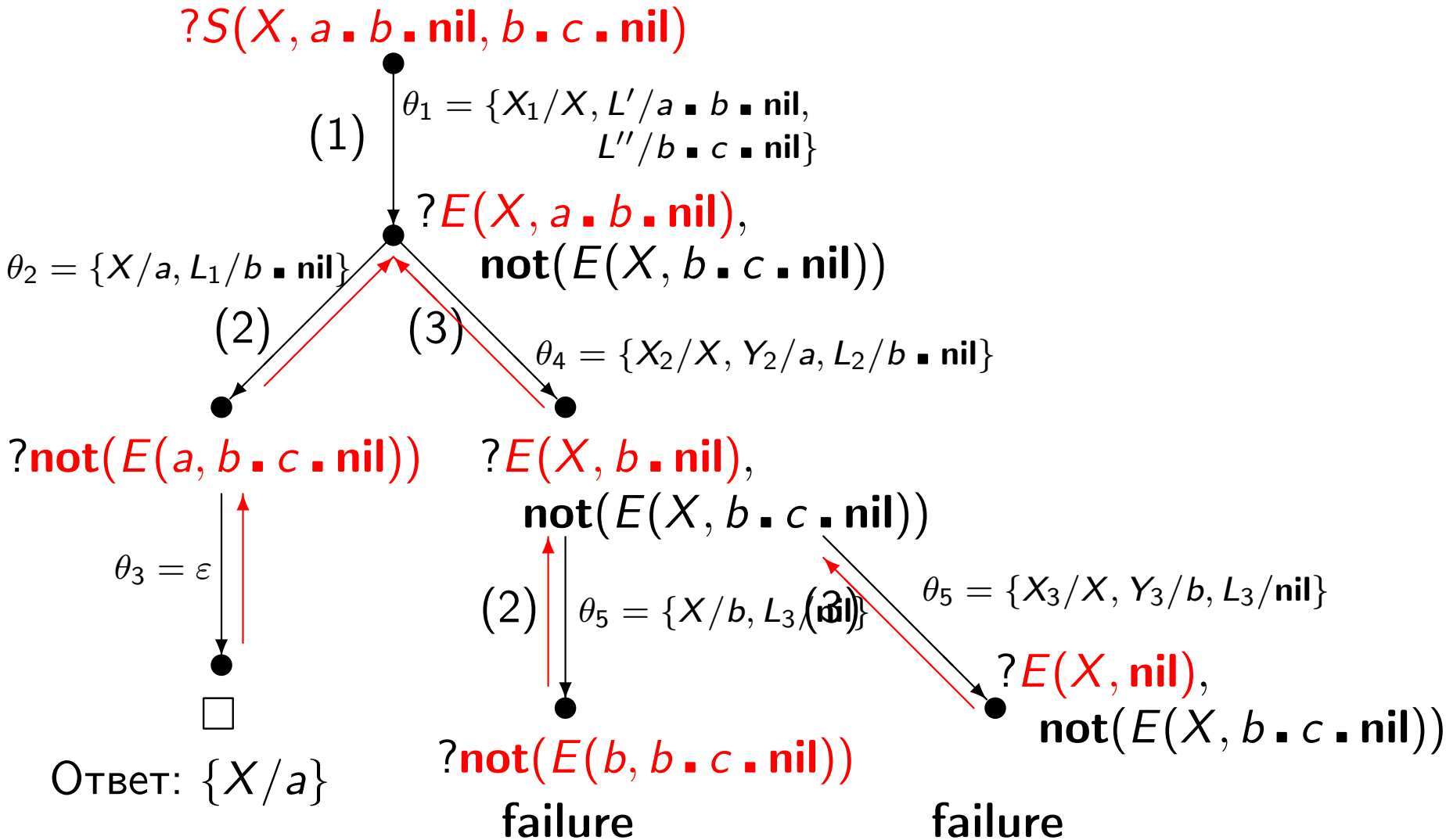
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''))$; (1)
$E(X, X \centerdot L) \leftarrow$; (2)
$E(X, Y \centerdot L) \leftarrow E(X, L)$; (3)



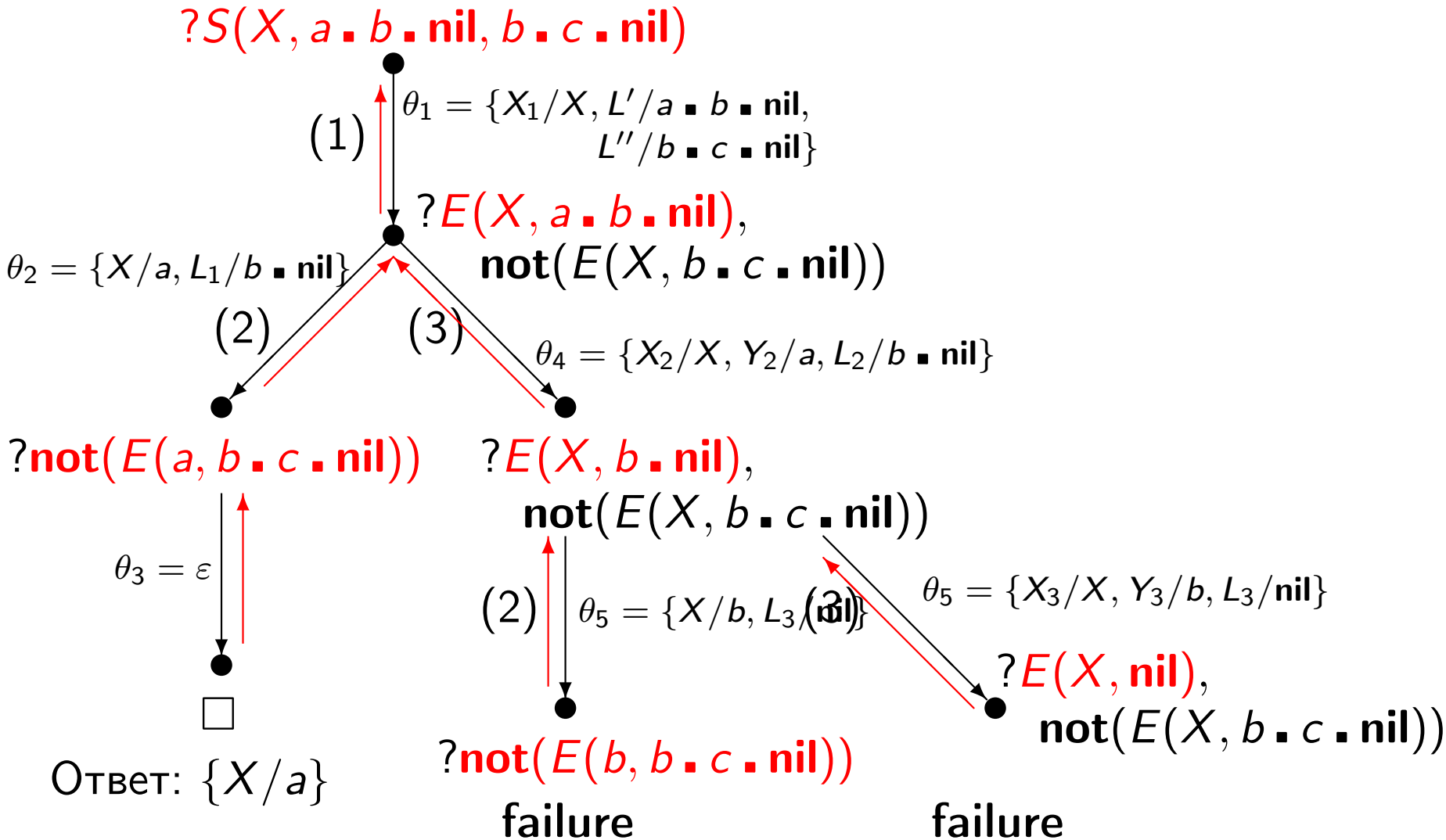
$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \cdot L) \leftarrow;$ (2)
$E(X, Y \cdot L) \leftarrow E(X, L);$ (3)



$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



$S(X, L', L'') \leftarrow E(X, L'), \mathbf{not}(E(X, L''));$ (1)
$E(X, X \centerdot L) \leftarrow;$ (2)
$E(X, Y \centerdot L) \leftarrow E(X, L);$ (3)



ОПЕРАТОР not

В том случае, когда построение дерева SLD-резолютивных вычислений проводится по стандартной стратегии, оператор **not** можно выразить через оператор отсечения **!**.

Для этого введем логическую константу **fail** — 0-местный предикат, имеющий постоянное значение *false*. С точки зрения операционной семантики, **fail** — это задача, которая не имеет решений. Тогда...

для обработки запроса ? **not**(C_0) в рамках SLD-резолютивного вывода с отсечением достаточно добавить к программе $\mathcal{P}$ два дополнительных программных утверждения

$$\begin{aligned} \textbf{not}(C_0) &\leftarrow C_0, \textbf{!}, \textbf{fail}; \\ \textbf{not}(C_0) &\leftarrow ; \end{aligned}$$

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Хорновские логические программы — это универсальная модель вычислений, в которой можно реализовать любой алгоритм. Однако для удобства пользования логическими программами к ним можно добавлять специальные встроенные функции и предикаты, операционная семантика которых определяется **вне рамок** SLD- резолютивного вывода.

Рассмотрим некоторые наиболее широко используемые встроенные средства логического программирования.

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат равенства $=$

Предикат равенства $t_1 = t_2$ — это 2-х местный предикат, предназначенный для унификации термов. Его операционная семантика задается следующими правилами:

$? \; t_1 = t_2$

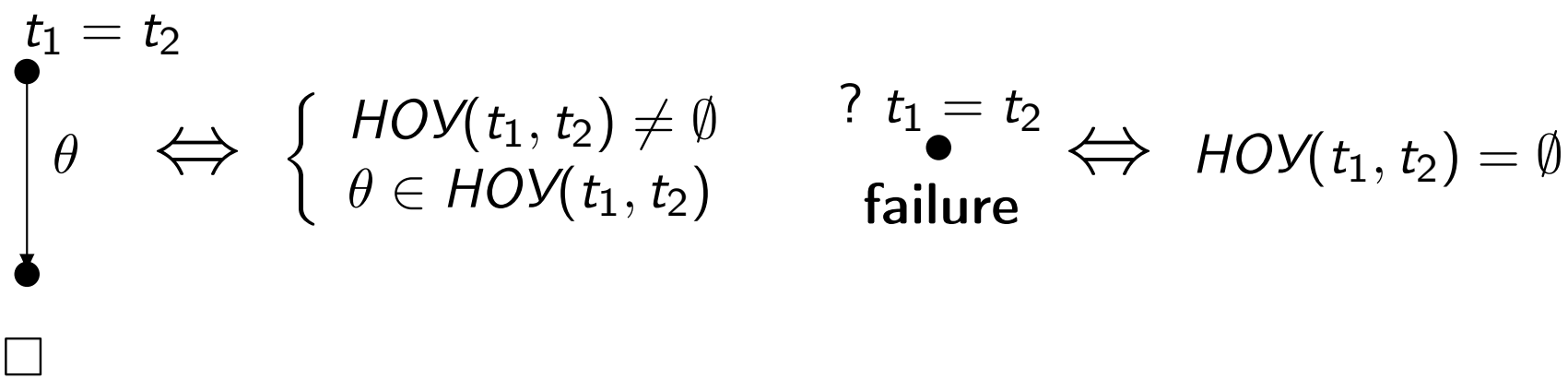
$$\theta \quad \Leftrightarrow \quad \begin{cases} HOU(t_1, t_2) \neq \emptyset \\ \theta \in HOU(t_1, t_2) \end{cases}$$

$\square$

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат равенства $=$

Предикат равенства $t_1 = t_2$ — это 2-х местный предикат, предназначенный для унификации термов. Его операционная семантика задается следующими правилами:

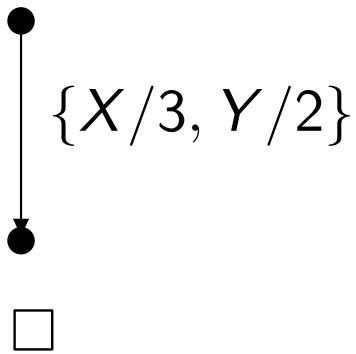


ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат равенства $=$

Пример

$? \; X + 2 = 3 + Y$

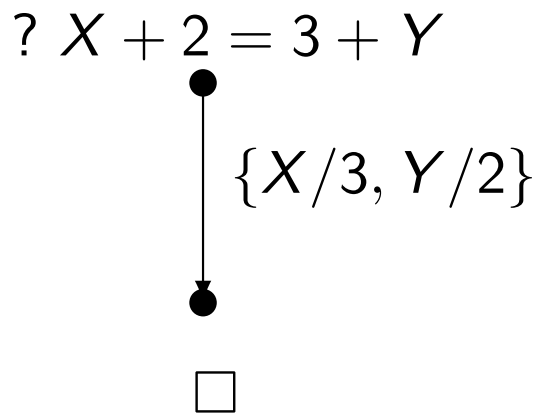


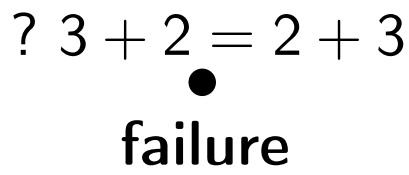
$\square$

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат равенства $=$

Пример



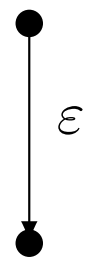


ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат тождества $==$

Предикат тождества $t_1 == t_2$ — это 2-х местный предикат, предназначенный для проверки тождественности (синтаксической идентичности) термов. Его операционная семантика задается следующими правилами:

$? \; t == t$



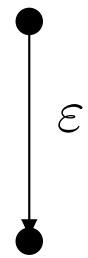
$\square$

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат тождества $==$

Предикат тождества $t_1 == t_2$ — это 2-х местный предикат, предназначенный для проверки тождественности (синтаксической идентичности) термов. Его операционная семантика задается следующими правилами:

$? \; t == t$



$\square$

$? \; t_1 == t_2$

failure $\Leftrightarrow$ t_1, t_2 — разные термы

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат тождества $==$

Пример

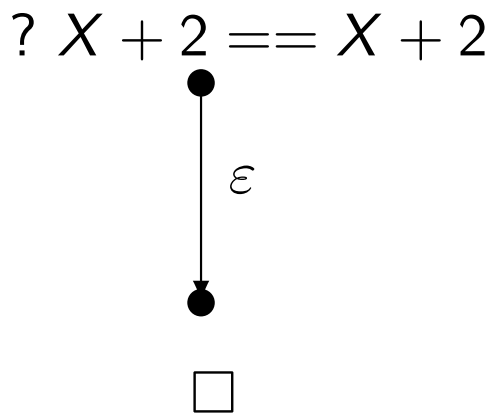
$? \; X + 2 == X + 2$

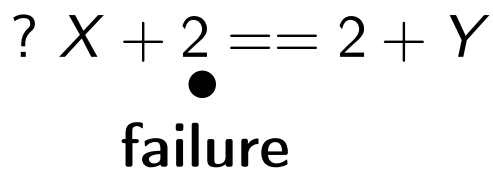
$\square$

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикат тождества ==

Пример





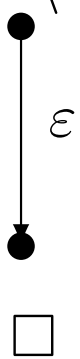
ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикаты неравенства $\backslash =$, $= \backslash =$

Предикаты неравенства $t_1 \backslash = t_2$, $t_1 = \backslash = t_2$ определяются при помощи оператора отрицания выражениями $\mathbf{not}(t_1 = t_2)$ и $\mathbf{not}(t_1 == t_2)$.

Пример

$? \; 3 + 2 \backslash = 2 + 3$



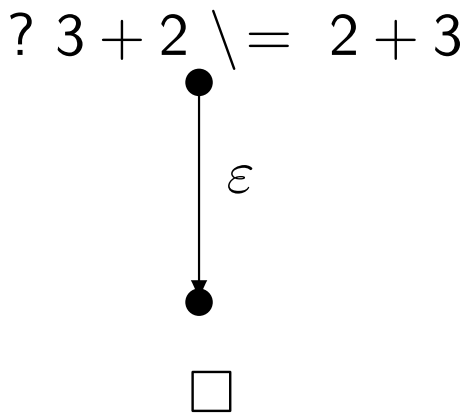
$\square$

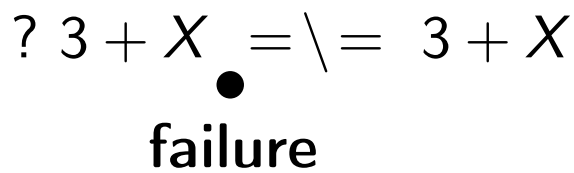
ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Предикаты неравенства $\backslash =$, $= \backslash =$

Предикаты неравенства $t_1 \backslash = \ t_2$, $t_1 \ = \backslash = t_2$ определяются при помощи оператора отрицания выражениями $\mathbf{not}(t_1 = t_2)$ и $\mathbf{not}(t_1 == t_2)$.

Пример





ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Типы данных

Для определения более сложных встроенных предикатов вводятся типы данных

integer, **real**, **boolean**, **char**, и т. д.

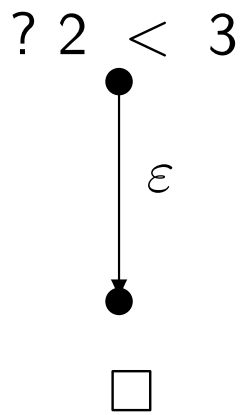
Каждый тип данных определяется множеством констант, обозначающих элементы этого типа данных. Например, **boolean** $= \{true, false\}$.

Тогда в каждом типе данных вводятся необходимые отношения, присущие элементам этого типа. Например, в типе данных **integer** можно ввести отношения сравнения $<$, $<=$, $>=$, $>$, и др.

ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Типы данных

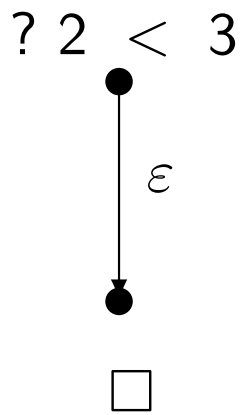
Пример

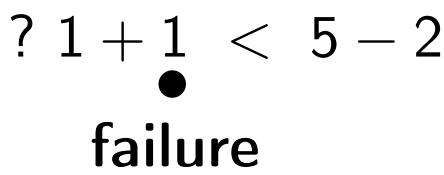


ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Типы данных

Пример



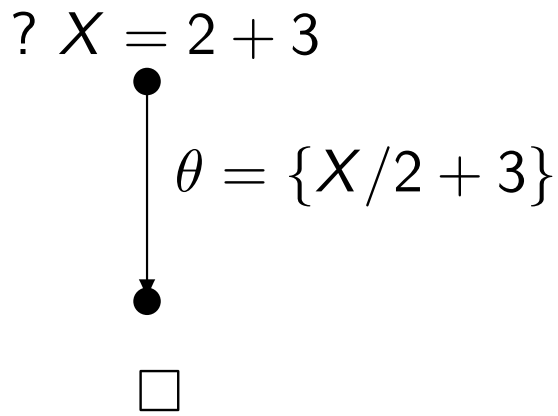


ВСТРОЕННЫЕ ФУНКЦИИ И ПРЕДИКАТЫ

Типы данных

Для введенных типов данных можно ввести встроенные функции (операции), присущие элементам этих типов
Например, в типе данных **integer** можно ввести двухместные функции (операции) $+$, $-$, $\times$, **div**, и др.

Однако чтобы вычисленные значения можно было передавать переменным, необходимо специальное средство. Предикат равенства $=$ для этой цели не годится, поскольку он занимается лишь унификацией термов.



ПРЕДИКАТ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ

Предикат вычисления значений **is** — это встроенный предикат, предназначенный для вычисления значений встроенных функций. Операционная семантика этого предиката задается следующими правилами.

Условимся использовать запись $val(t)$ для обозначения значение терма t, составленного из встроенных функций и констант. Тогда...

? t_1 **is** t_2

$\theta \quad \Leftrightarrow \quad \begin{cases} t_1 \in Var, \\ \text{определено } val(t_2) \end{cases}$

$\square \qquad \theta = \{t_1 / val(t_2)\}$

ПРЕДИКАТ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ

Предикат вычисления значений **is** — это встроенный предикат, предназначенный для вычисления значений встроенных функций. Операционная семантика этого предиката задается следующими правилами.

Условимся использовать запись $val(t)$ для обозначения значение терма t, составленного из встроенных функций и констант. Тогда...

? t_1 **is** t_2

$\theta \quad \Leftrightarrow \quad \begin{cases} t_1 \in Var, \\ \text{определено } val(t_2) \end{cases}$

$\square \qquad \theta = \{t_1 / val(t_2)\}$

ПРЕДИКАТ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ

Предикат вычисления значений **is** — это встроенный предикат, предназначенный для вычисления значений встроенных функций. Операционная семантика этого предиката задается следующими правилами.

Условимся использовать запись $val(t)$ для обозначения значение терма t, составленного из встроенных функций и констант. Тогда...

? t_1 **is** t_2

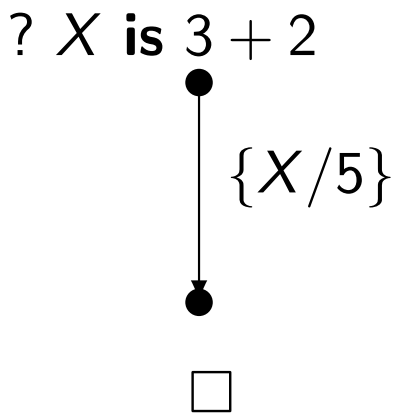
$\theta \quad \Leftrightarrow \quad \begin{cases} t_1 \in Var, \\ \text{определено } val(t_2) \end{cases}$

? t_1 **is** t_2 **failure** $\Leftrightarrow$ в остальных случаях

$\square \qquad \theta = \{t_1/val(t_2)\}$

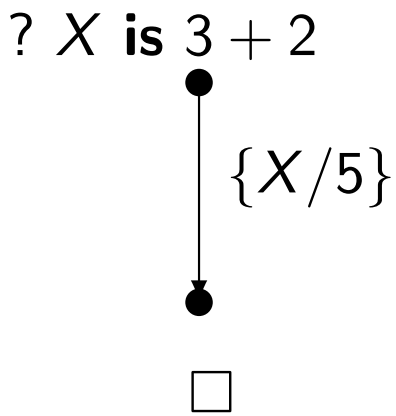
ПРЕДИКАТ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ

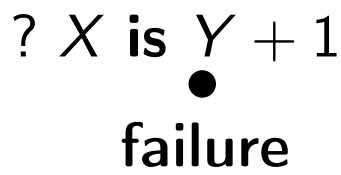
Пример



ПРЕДИКАТ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ

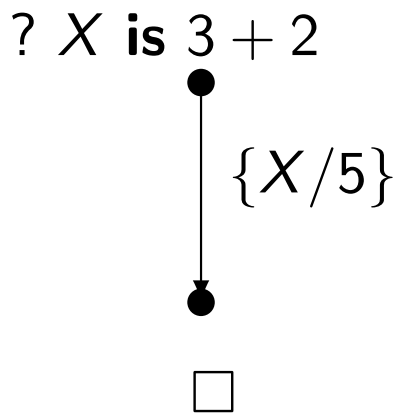
Пример

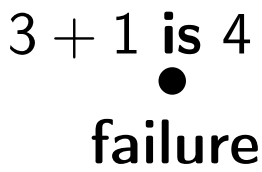




ПРЕДИКАТ ВЫЧИСЛЕНИЯ ЗНАЧЕНИЙ

Пример





МОДИФИКАЦИЯ БАЗ ДАННЫХ

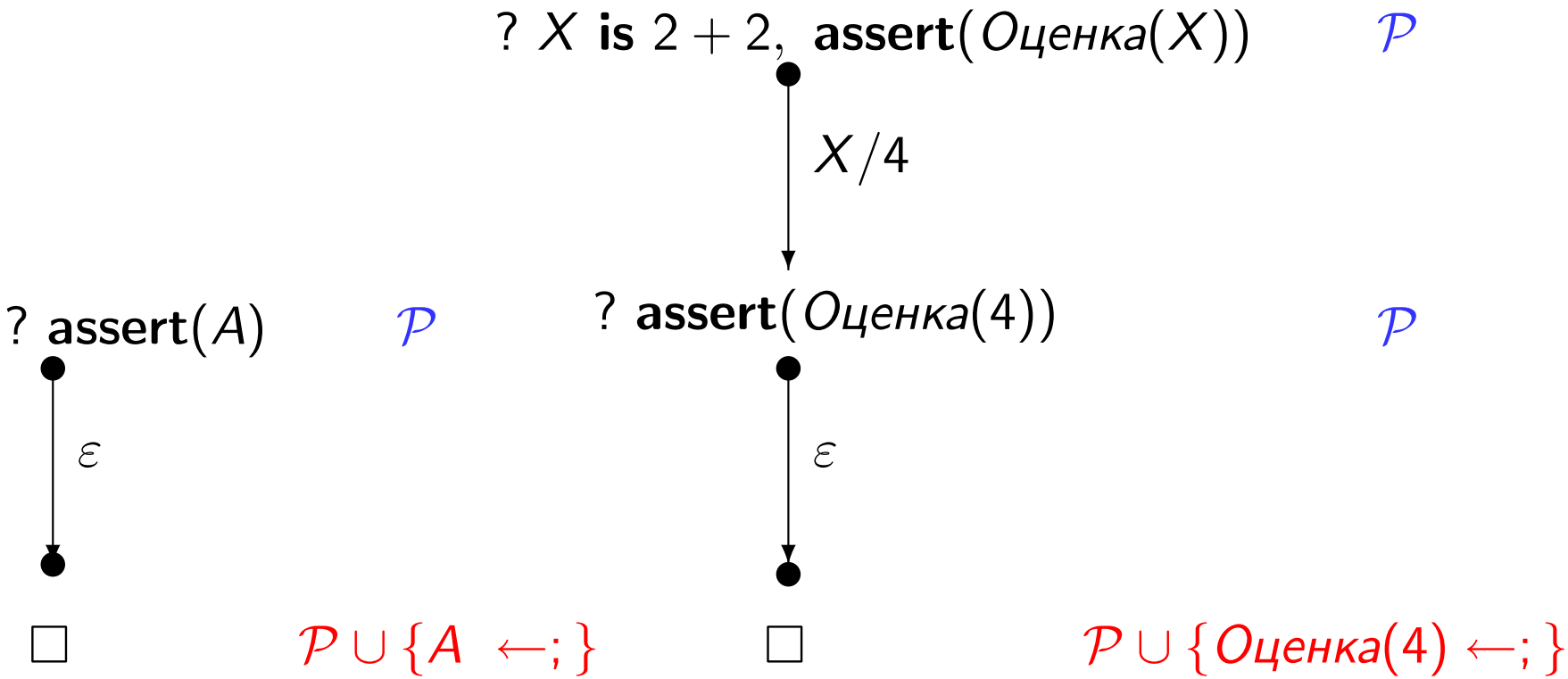
Обычно логическая программа $\mathcal{P}$ разделяется на две части — систему правил $\mathcal{P}_{clauses}$ и базу данных $\mathcal{P}_{database}$. База данных состоит из всех основных фактов программы.

Система правил $\mathcal{P}_{clauses}$ представляет, по сути дела, алгоритм решения задачи, и его неразумно изменять по ходу решения задачи.

А вот базу данных $\mathcal{P}_{database}$ по ходу вычисления можно модиифцировать. И для этой цели в системах логического программирования есть специальные встроенные операторы пополнения и сокращения базы данных.

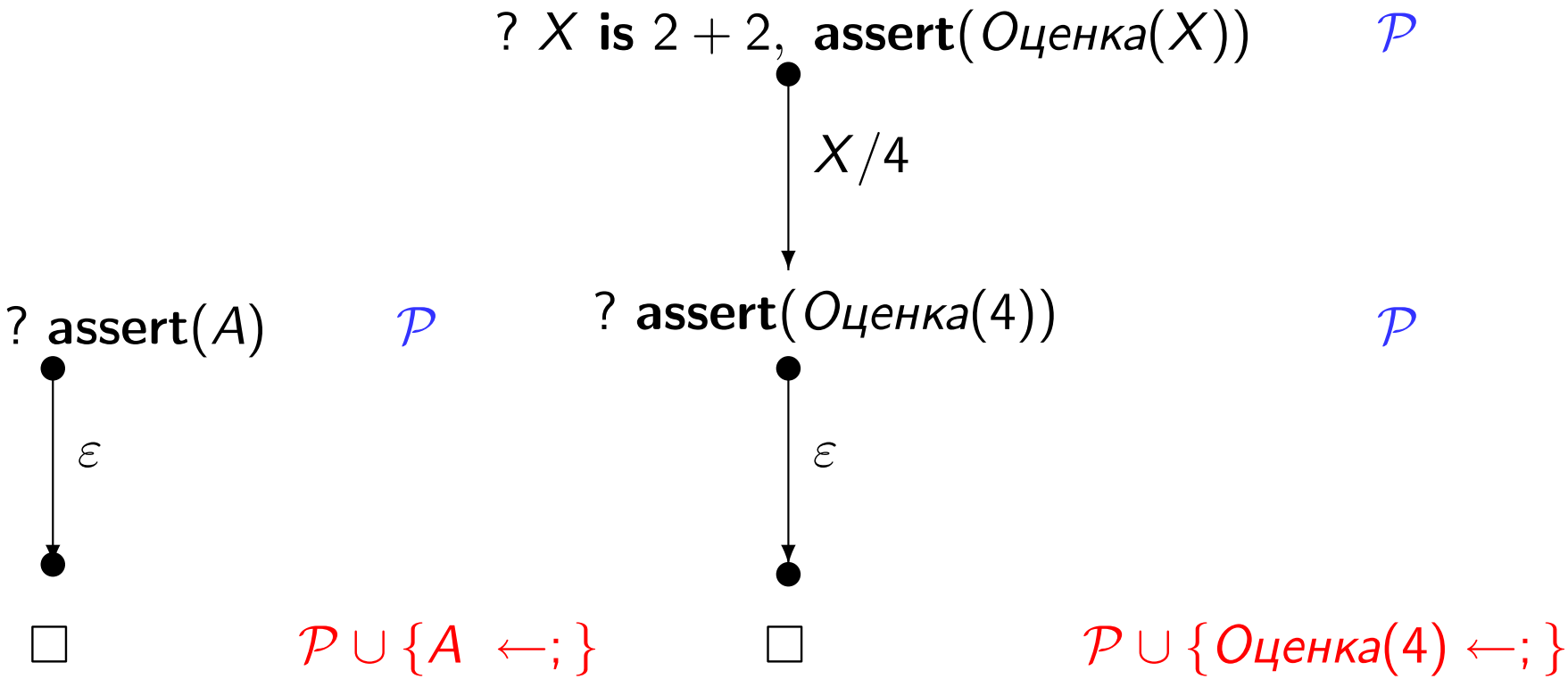
МОДИФИКАЦИЯ БАЗ ДАННЫХ

Оператор пополнения базы данных **assert**(A), где A — атом, добавляет к базе данных факт $A \leftarrow$;



МОДИФИКАЦИЯ БАЗ ДАННЫХ

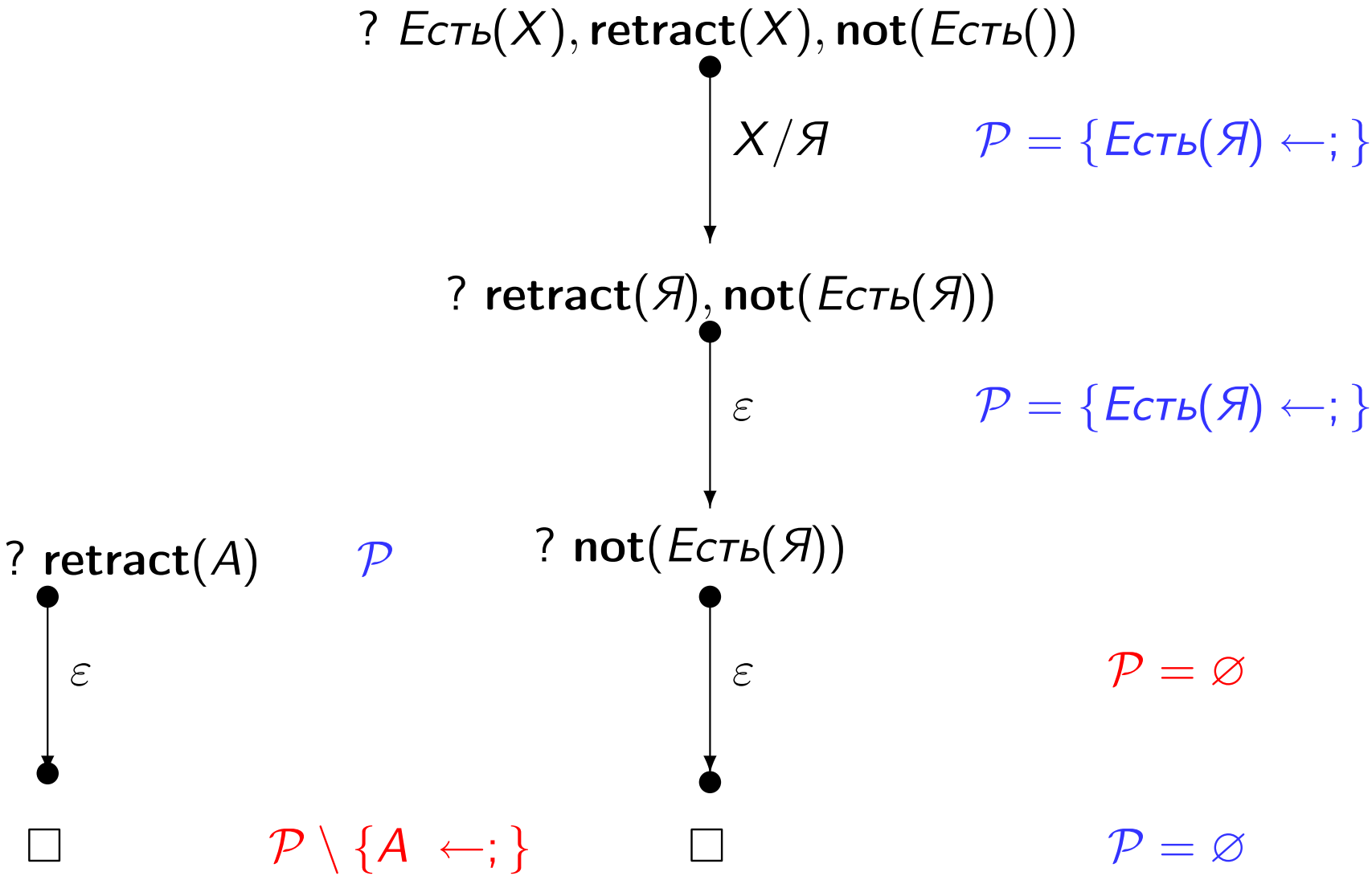
Оператор пополнения базы данных **assert**(A), где A — атом, добавляет к базе данных факт $A \leftarrow$;



Поскольку в логических программах правила и факты упорядочены, нужны два варианта оператора пополнения баз данных: **asserta**(A) и **assertz**(A).

МОДИФИКАЦИЯ БАЗ ДАННЫХ

Оператор сокращения базы данных **retract**(A), где A — атом, удаляет из базы данных факт $A \leftarrow$;



МОДИФИКАЦИЯ БАЗ ДАННЫХ

Творческая задача

А что если разрешить в теле правила использовать наряду с атомами также и программные утверждения? Как то, например,

$$A_0 \leftarrow A_1, \ldots, A_i, (B_0 \leftarrow B_1, \ldots, B_m), A_{i+1}, \ldots, A_n;$$

или $$A_0 \leftarrow A_1, \ldots, A_i, (B_0 \leftarrow), A_{i+1}, \ldots, A_n;$$

1. Как определить разумную декларативную семантику таких «составных» правил?
2. Как определить адекватную операционную семантику таких правил?
3. Можно ли определить адекватную операционную семантику «составных» правил на основе резолютивного вывода?

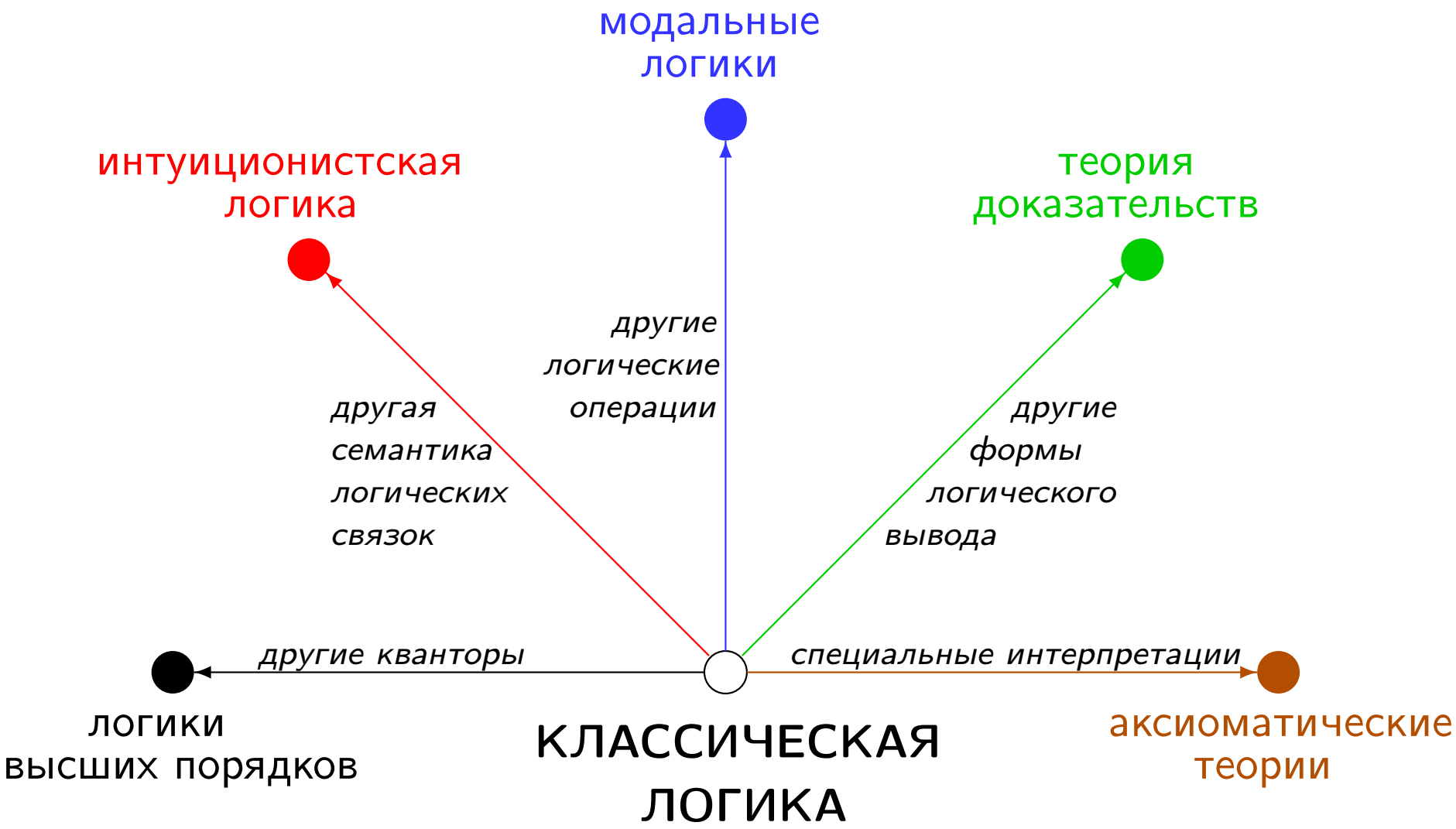
КОНЕЦ ЛЕКЦИИ 17.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 18-19.

Интуиционистская логика. Модальные логики.


модальные
логики
интуиционистская
логика
теория
доказательств
другие
логические
операции
другая
семантика
логических
связок
другие
формы
логического
вывода
другие кванторы
специальные интерпретации
логики
высших порядков
КЛАССИЧЕСКАЯ
ЛОГИКА
аксиоматические
теории

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Интуиционизм — это философское течение в математике, возникшее в начале 20 века как критический отклик на неограниченное применение формальных логических методов в математике, приводящее к парадоксам (антиномиям).
По мнению интуиционистов (Брауэр, Вейль, Пуанкаре), парадоксы возникают в связи с тем, что законы логики, справедливые для конечных множеств, безосновательно переносятся на бесконечные множества.

Не все математические утверждения, верные для конечных множеств, остаются справедливыми и для бесконечных множеств. Например, для конечных множеств верен принцип Архимеда «**Часть всегда меньше целого**», а для бесконечных множеств — нет.

Вполне возможно, что не все законы классической (аристотелевой) логики допускают неограниченное и безоговорочное использование в математике.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Например, рассмотрим одну широко распространенную схему доказательства.

Доказать: Если выполнены условия A, то $\exists x\ P(x)$.

Схема доказательства: Предположим противное, т. е. $\forall x\ \neg P(x)$.
Тогда ...(фа-фа, ля-ля)..., что противоречит условиям A.
Значит, предположение $\forall x\ \neg P(x)$ неверно, и поэтому $\exists x\ P(x)$.
QED

Все хорошо, но где же та x, для которой верно $P(x)$?
Из такого доказательства это значение извлечь невозможно.

Но тогда, по мнению интуиционистов, это не доказательство, а словоблудие.

Чтобы исключить доказательства такого рода, нужно пересмотреть семантику логических связок и кванторов.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Семантика Колмогорова–Брауэра–Гейтинга

Попробуем взглянуть на логические формулы как на утверждения о разрешимости математических задач.

Каждая атомарная формула A будет обозначать некоторую задачу. Истинность A будет означает, что задача имеет решение, и это решение можно предъявить. Ложность A будет означать, что задача решения не имеет.

Логические связки позволяют конструировать из простых задач составные задачи.

Оценим, как (не)разрешимость составных задач зависит от (не)разрешимости простых задач.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Семантика Колмогорова–Брауэра–Гейтинга

$\varphi \,\&\, \psi$: Решить обе задачи φ и ψ и предъявить решение;

$\varphi \vee \psi$: Выбрать одну из двух задач φ и ψ, решить выбранную задачу и предъявить решение;

$\varphi \to \psi$: Показать, что решение задачи ψ сводится к решению задачи φ, т. е. предъявить способ, который позволяет, располагая решением задачи φ, построить решение задачи ψ;

$\neg \varphi$: Доказать, что задача φ не имеет решения.

Законами интуиционистской логики считаются только те формулы, которые соответствуют описаниям составных задач, имеющих решение при любых условиях.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Законы интуиционистской логики

- $P \to P$ — каждую задачу можно свести к ней самой;
- $(P \to Q)\&(Q \to R) \to (P \to R)$ — чтобы свести задача R к задаче P достаточно найти задачу Q, к которой можно свести задачу R, и которую, в свою очередь, можно свести к задаче P;
- $P \to \neg\neg P$ — чтобы убедиться в том, что не существует доказательства неразрешимости задачи P, достаточно найти решение задачи P;
- $(\neg P \vee \neg Q) \to \neg(P\&Q)$ — чтобы показать, что обе задачи P и Q нельзя решить одновременно, достаточно выбрать одну из этих задач и показать, что она неразрешима.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Формулы, не являющиеся законами интуиционистской логики

- $\neg\neg P \to P$ — если вы можете обосновать, что нельзя построить доказательства неразрешимости задачи P, то этого еще недостаточно, чтобы получить решение самой задачи P;
- $P \vee \neg P$ — неправда, что для любой задачи можно либо получить решение, либо доказать, что никакого решения не существует;
- $\neg(P\&Q) \to (\neg P \vee \neg Q)$ — если можно доказать, что обе задачи P и Q нельзя решить одновременно, то это не дает основания считать, что хотя бы одна из них является неразрешимой.

Да как же это так?

Уж не скрывается ли здесь простая игра слов?

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Попробуем строго определить семантику утверждений, касающихся разрешимости задач.
Истинность формул оценивается в интерпретациях. Поскольку задачи решают люди, в качестве интерпретаций могут выступать способности людей решать задачи.
Но эти способности у людей со временем изменяются. Значит, интерпретации должны быть динамическими .
Рассмотрим модель идеального математика (**Dutch Mathematician**), который

- может пребывать в разных состояниях знания и переходить из одних состояний знания в другие;
- в каждом состоянии знания он точно знает, какие из элементарных задач он умеет решать, а какие нет;
- не утрачивает навыков в решении задач при переходе из одного состояния знания в другое.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Определение (модель Крипке)

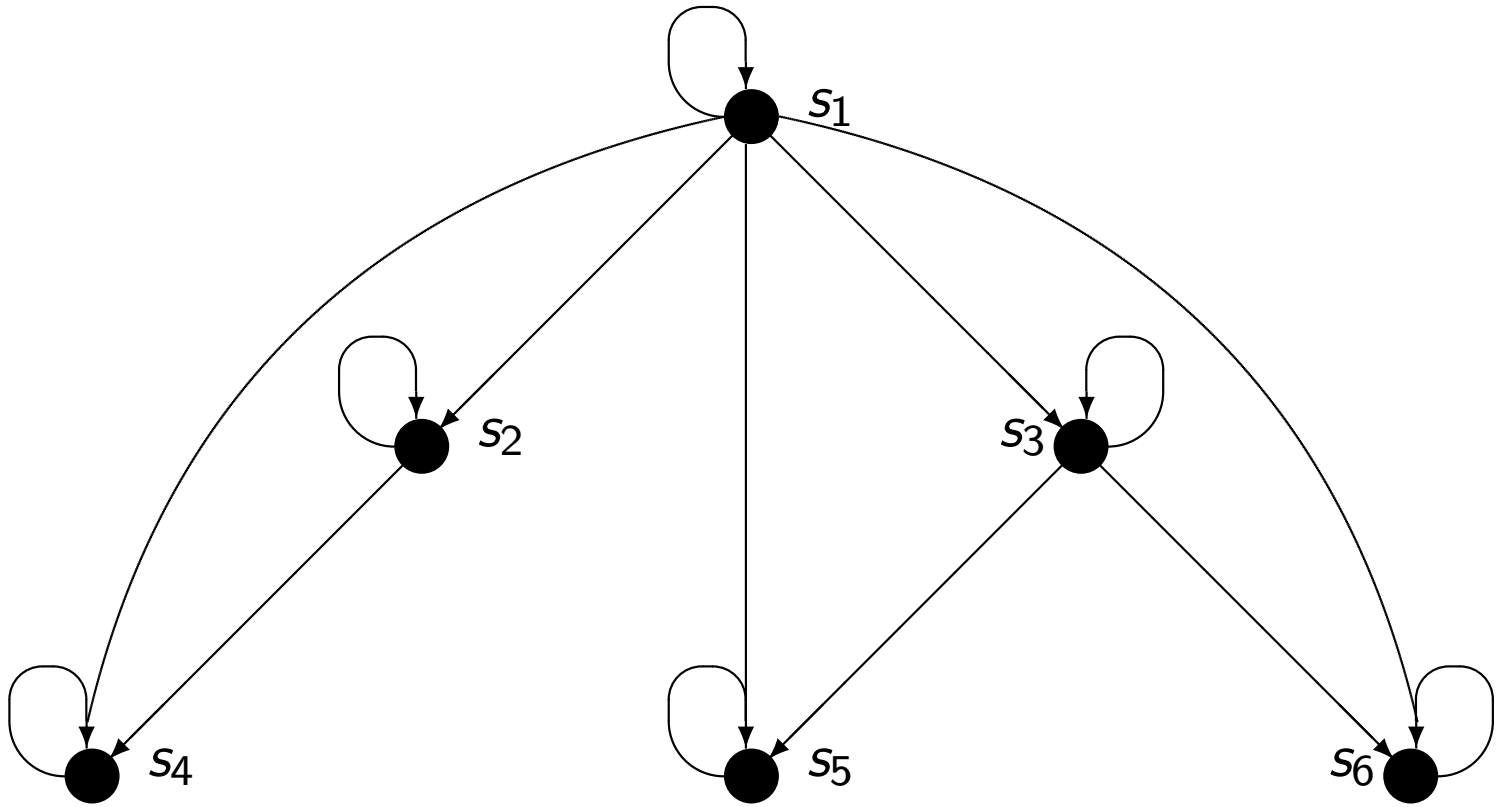
Пусть $\mathcal{P} = \{P_1, P_2, \ldots, P_n, \ldots\}$ — множество атомарных формул (названия задач).

Интуиционистская интерпретация — это реляционная система $I = \langle S, \mathbf{R}, \xi \rangle$, в которой

1. $S \neq \emptyset$ — множество состояний (состояний знания);
2. $\mathbf{R} \subseteq S \times S$ — отношение переходов на S, которое является отношением нестрогого частичного порядка:
   рефлексивное $R(s, s)$;
   транзитивное $R(s_1, s_2) \& R(s_2, s_3) \Rightarrow R(s_1, s_3)$;
   антисимметричное $R(s_1, s_2) \& R(s_2, s_1) \Rightarrow s_1 = s_2$;
3. $\xi : S \times \mathcal{P} \to \{\mathbf{true}, \mathbf{false}\}$ — оценка атомарных формул, удовлетворяющая условию монотонности:
   $R(s_1, s_2)\ \&\ \xi(P, s_1) = \mathbf{true} \ \Rightarrow\ \xi(P, s_2) = \mathbf{true}$.

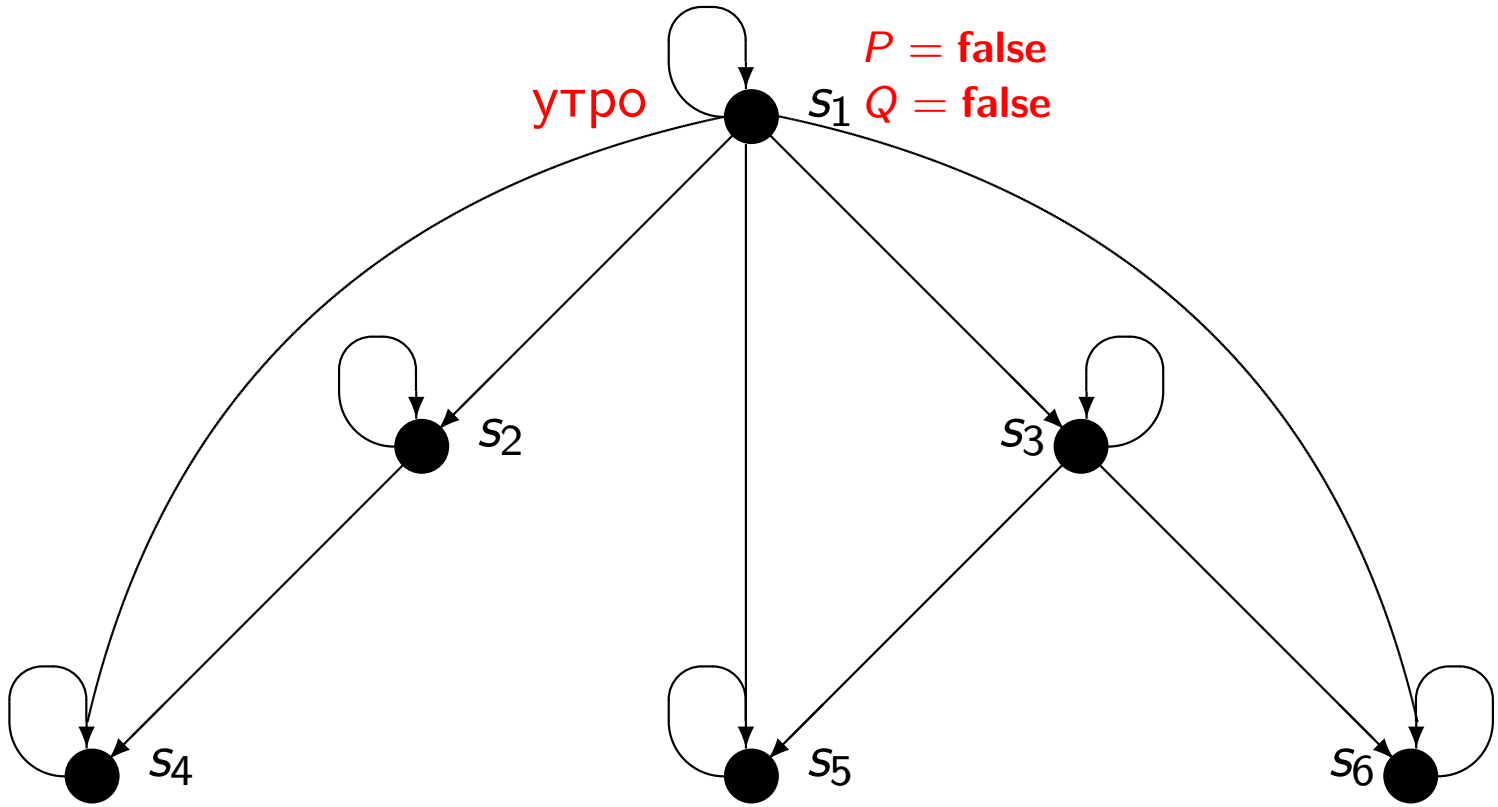
ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример интуиционистской интерпретации



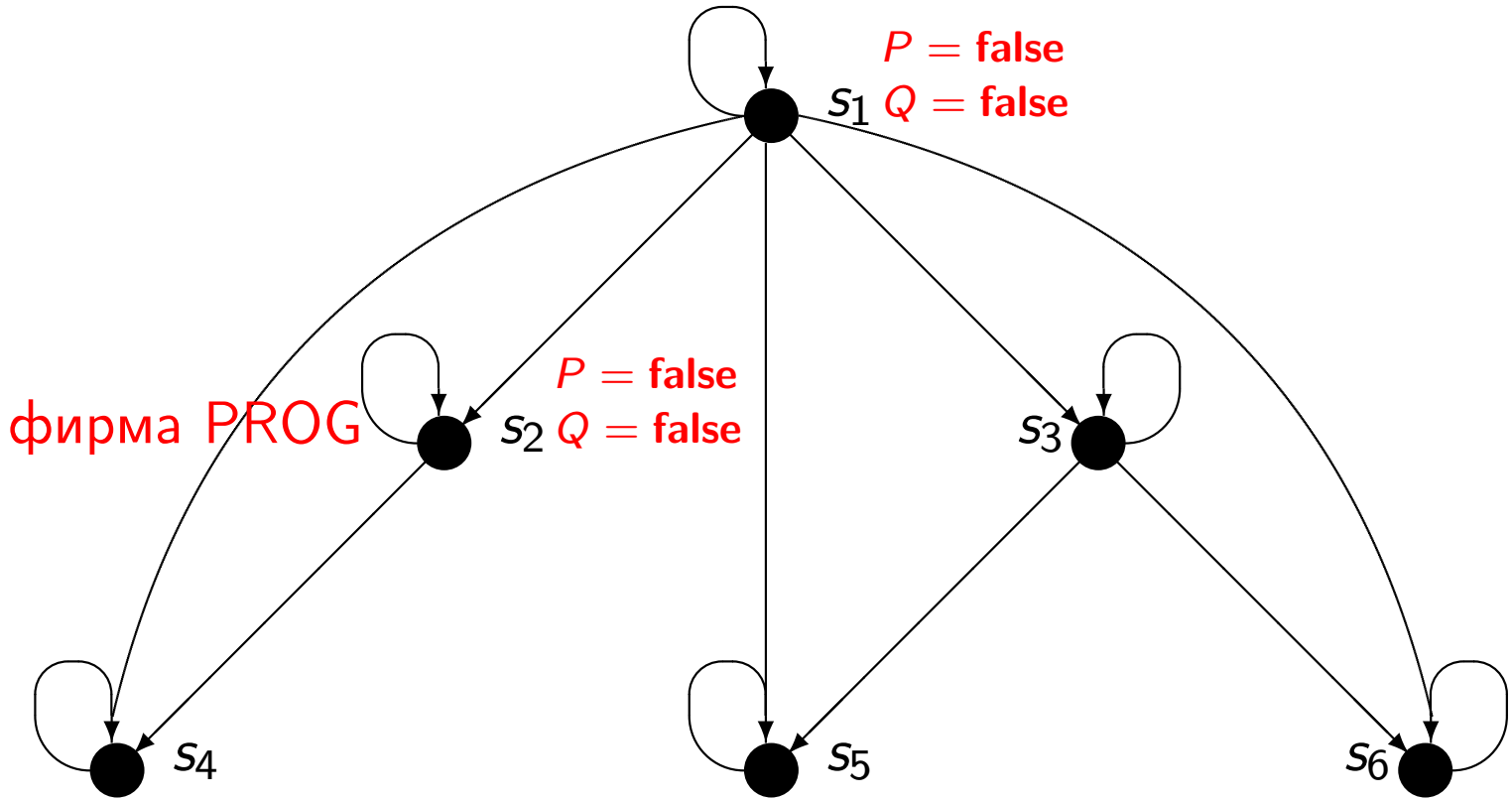
ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример интуиционистской интерпретации



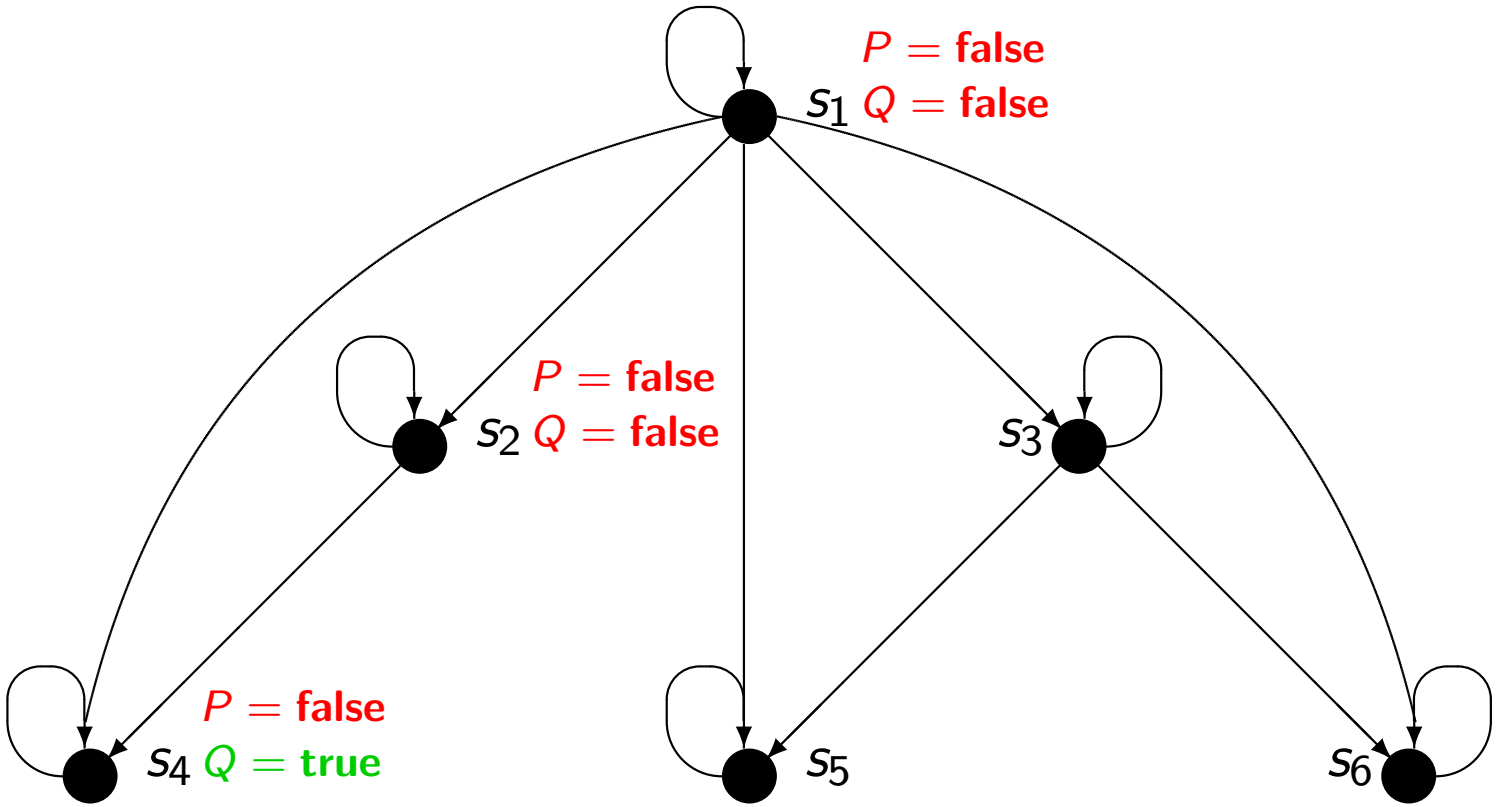
ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример интуиционистской интерпретации



ИНТУИЦИОНИСТСКАЯ ЛОГИКА

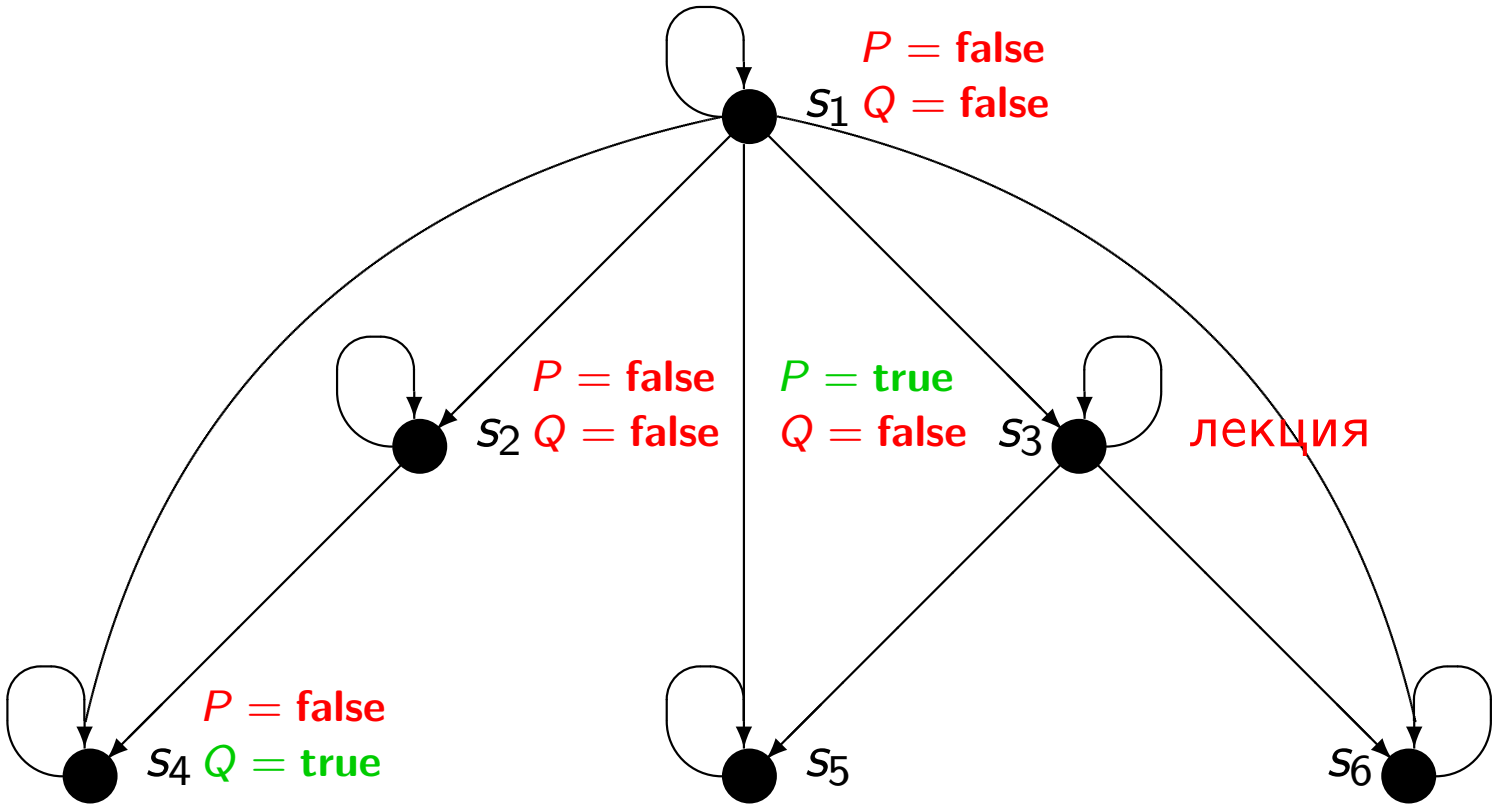
Пример интуиционистской интерпретации



экзамен

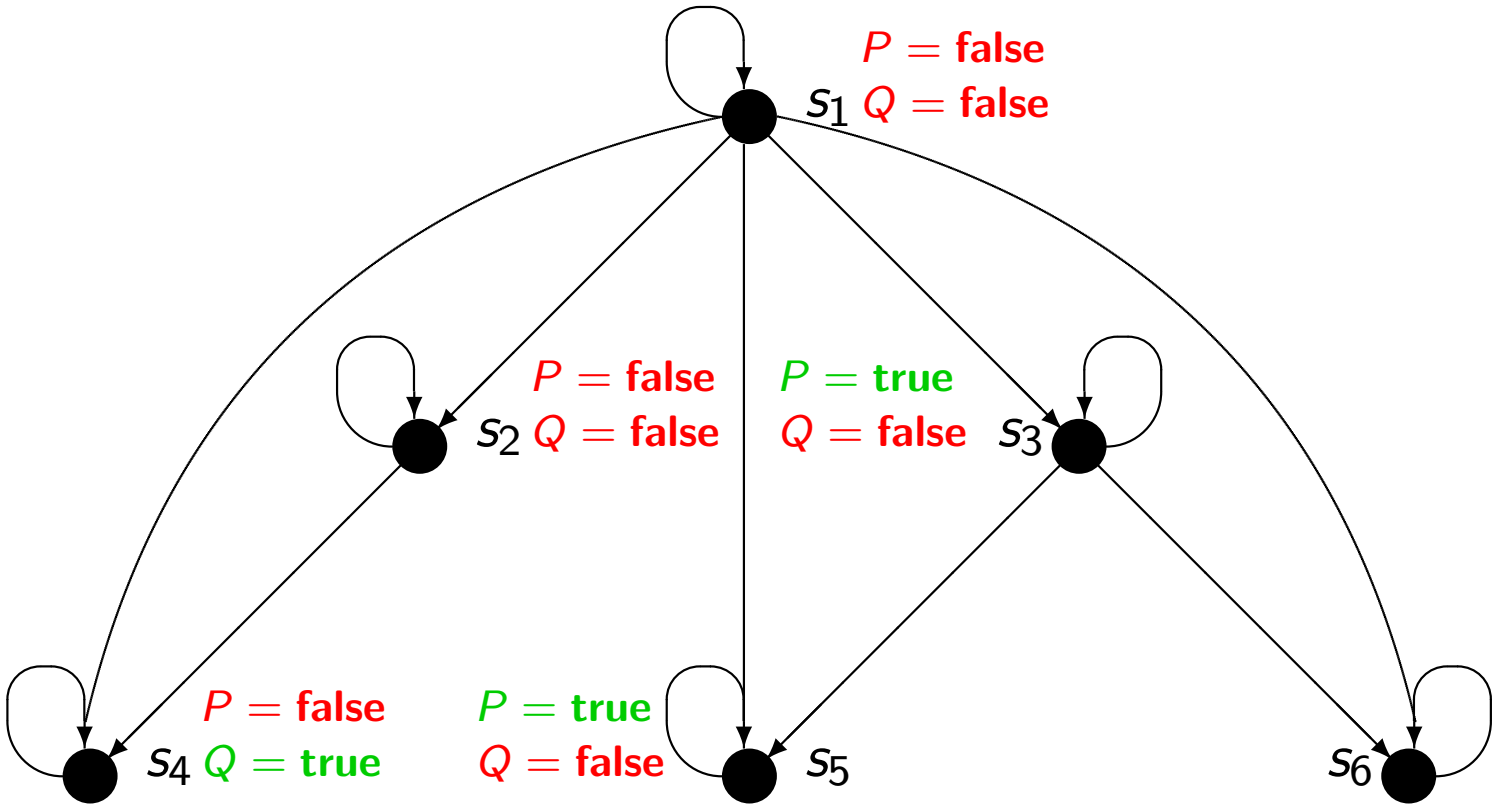
ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример интуиционистской интерпретации



ИНТУИЦИОНИСТСКАЯ ЛОГИКА

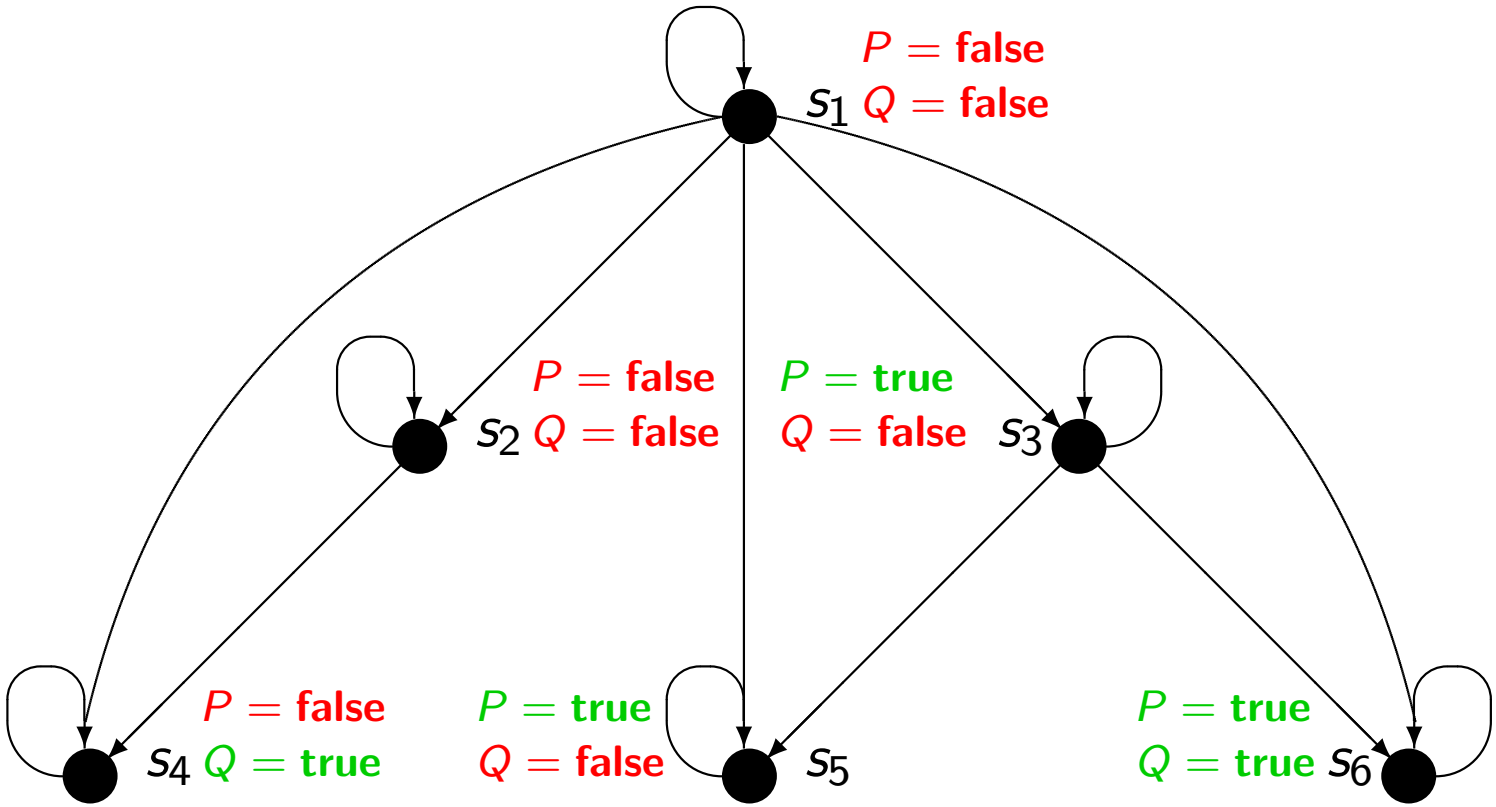
Пример интуиционистской интерпретации



экзамен

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

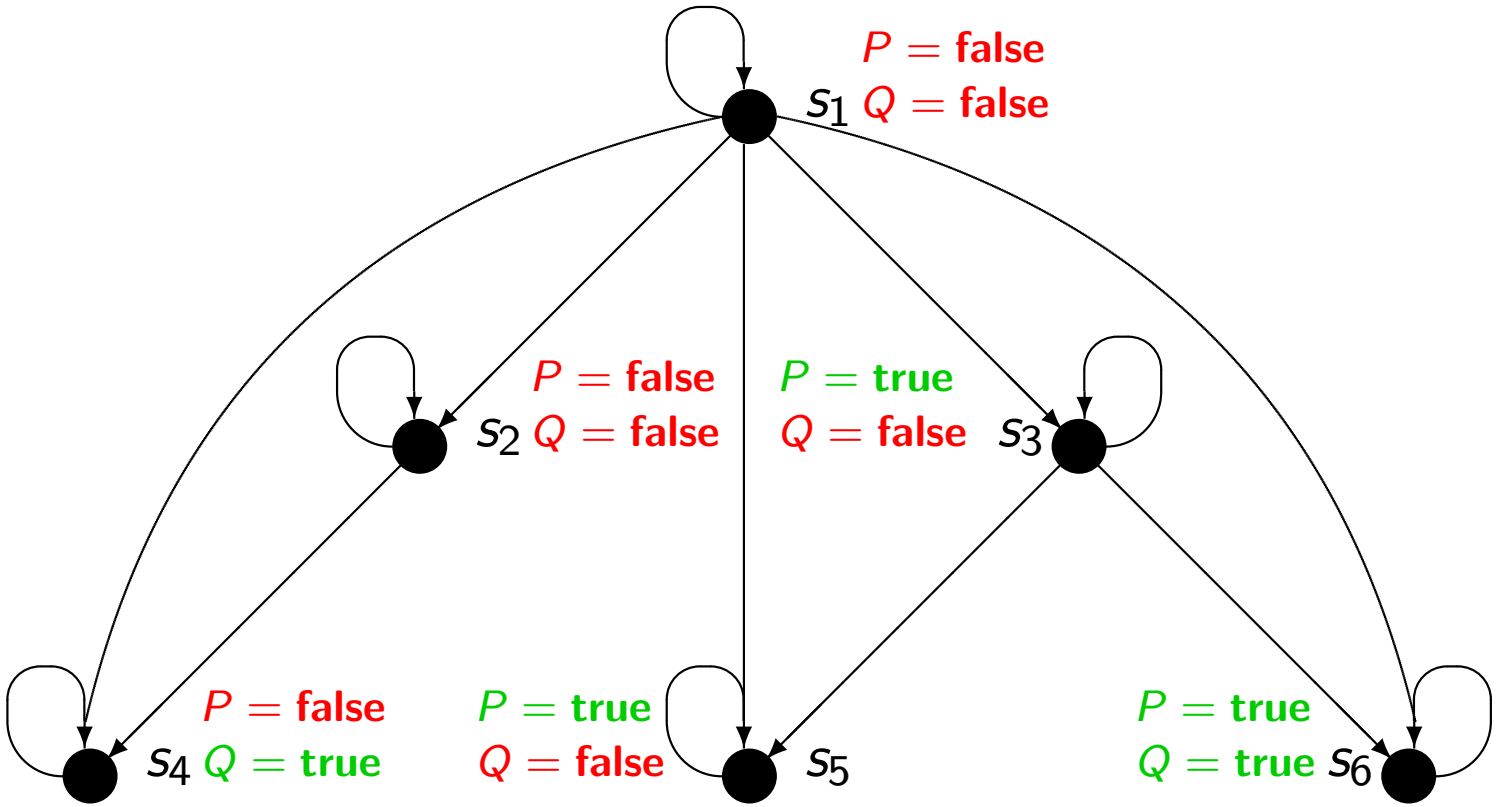
Пример интуиционистской интерпретации



экзамен

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример интуиционистской интерпретации



ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Определение (семантика Крипке)

Пусть $I = \langle S, \mathbf{R}, \xi \rangle$ — интуиционистская интерпретация. Тогда отношение выполнимости $I, s \models_{\mathcal{I}} \varphi$ формулы φ в состоянии s интерпретации I определяется так:

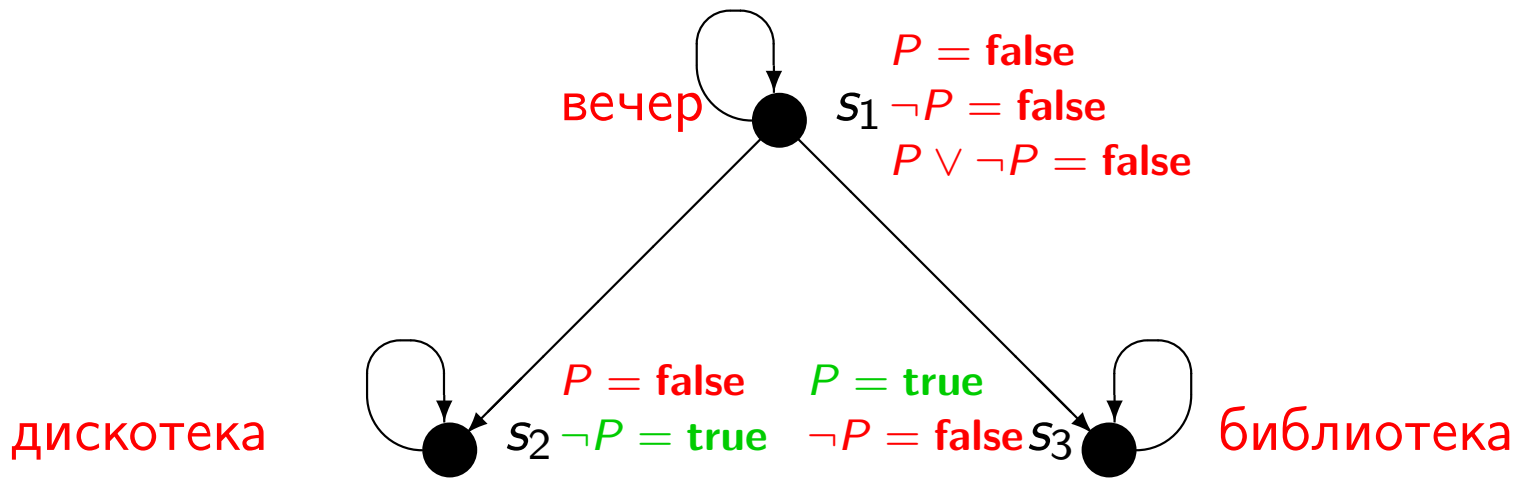
1. если $\varphi = P \in \mathcal{P}$, то $I, s \models_{\mathcal{I}} \varphi \iff \xi(s, P) = \mathbf{true}$;
2. $I, s \models_{\mathcal{I}} \varphi_1 \& \varphi_2 \iff I, s \models_{\mathcal{I}} \varphi_1$ и $I, s \models_{\mathcal{I}} \varphi_2$;
3. $I, s \models_{\mathcal{I}} \varphi_1 \vee \varphi_2 \iff I, s \models_{\mathcal{I}} \varphi_1$ или $I, s \models_{\mathcal{I}} \varphi_2$;
4. $I, s \models_{\mathcal{I}} \varphi_1 \to \varphi_2 \iff$ для любого состояния s', если $(s, s') \in \mathbf{R}$ и $I, s' \models_{\mathcal{I}} \varphi_1$, то $I, s' \models_{\mathcal{I}} \varphi_2$;
5. $I, s \models_{\mathcal{I}} \neg \varphi_1 \iff$ для любого состояния s', если $(s, s') \in \mathbf{R}$, то $I, s' \not\models_{\mathcal{I}} \varphi_1$.

Формула φ называется интуиционистски общезначимой (законом интуиционистской логики), если для любой интерпретации I и для любого состояния s верно $I, s \models_{\mathcal{I}} \varphi$.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример необщезначимой формулы

$$\not\models_{\mathcal{I}} P \vee \neg P$$



ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Другие необщезначимые формулы

Докажите самостоятельно, выбрав подходящую интерпретацию (контрмодель) I,

$$\not\models_{\mathcal{I}} \neg\neg P \to P$$

$$\not\models_{\mathcal{I}} \neg(P \& Q) \to (\neg P \vee \neg Q)$$

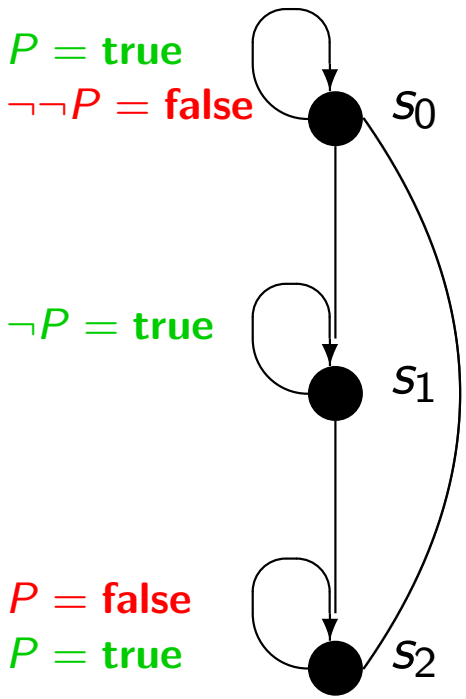
$$\not\models_{\mathcal{I}} \neg(P \vee Q) \to (\neg P \& \neg Q)$$

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Пример общезначимой формулы

$$\models_{\mathcal{I}} P \to \neg\neg P$$

От противного. Допустим, что $I, s_0 \not\models P \to \neg\neg P$. Тогда



Полученное противоречие свидетельствует о невозможности построения контрмодели для формулы $P \to \neg\neg P$.

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Другие общезначимые формулы

Докажите самостоятельно интуиционистскую общезначимость следующих формул

$$\models_{\mathcal{I}} \neg\neg\neg P \to \neg P$$

$$\models_{\mathcal{I}} (\neg P \vee \neg Q) \to \neg(P \& Q)$$

$$\models_{\mathcal{I}} \neg P \vee \neg\neg P$$

ИНТУИЦИОНИСТСКАЯ ЛОГИКА

Некоторые особенности интуиционистской логики

Теорема 1

$\models_{\mathcal{I}} \varphi \implies \models_{\mathcal{C}} \varphi$

Теорема 2 (дизъюнктивное свойство)

$\models_{\mathcal{I}} \varphi \vee \psi \iff \models_{\mathcal{I}} \varphi$ или $\models_{\mathcal{I}} \psi$

Теорема 3 (экзистенциальное свойство)

$$\models_{\mathcal{I}} \forall x_1 \ldots \forall x_1 \exists y \varphi(x_1, \ldots, x_n, y)$$
$$\iff$$
существует такой терм $t(x_1, \ldots, x_n)$, что
$$\models_{\mathcal{I}} \varphi(x_1, \ldots, x_n, t(x_1, \ldots, x_n))$$

$t(x_1, \ldots, x_n)$ — это программа решения задачи φ.

Это называется «изоморфизмом Карри–Ховарда».

МОДАЛЬНЫЕ ЛОГИКИ

Зимой идет снег

Зимой всегда идет снег

Зимой иногда идет снег

Это разные высказывания. И поэтому они должны быть записаны разными формулами.

Эти высказывания по смыслу связаны друг с другом. И это должно быть отражено в формулах. Поскольку высказывания отличаются лишь словами **всегда**, **иногда** (модальности времени), нужно ввести какие-то логические конструкции для выражения этих модальностей.

Может быть для этой цели пригодны кванторы?

МОДАЛЬНЫЕ ЛОГИКИ

Студенты посещают лекции

Студенты обязаны посещать лекции

Студенты имеют право посещать лекции

обязан, **имею право** — деонтические модальности .

А будут ли пригодны кванторы в этом случае для выражения модальностей?

МОДАЛЬНЫЕ ЛОГИКИ

Задача имеет решение

Известно, что задача имеет решение

Можно допустить, что задача имеет решение

знаю, **предполагаю** — эпистемические модальности.

А как быть здесь?

МОДАЛЬНЫЕ ЛОГИКИ

Модальности (в естественном языке они, как правило, представлены наречиями или служебными глаголами) выражают различные оттенки истинности (уверенность, необходимость, доказуемость, осведомленность и др.).

Эти оттенки можно классифицировать:

Модальности необходимого	Модальности возможного
необходимо	возможно
обязательно	не исключено
всегда	иногда
должна	имею право
знаю	предполагаю
доказуемо	непротиворечиво
$\Box$	$\Diamond$

МОДАЛЬНЫЕ ЛОГИКИ

Синтаксис модальных формул

Расширим синтаксис классической логики предикатов, введя два логических оператора
$\Box$ (модальность необходимого) и
$\Diamond$ (модальность возможного),
при помощи которых разрешается строить формулы следующего вида:

$(\Box\varphi)$ «необходимо φ»,

$(\Diamond\varphi)$ «возможно φ».

Во избежание большого количества скобок, будем считать, что модальные операторы имеют такой же приоритет, что и кванторы.

МОДАЛЬНЫЕ ЛОГИКИ

Семантика модальных формул многообразна и непроста. Рассмотрим

Пример

Верно ли, что формула $\Box\varphi \to \varphi$ — это закон модальной логики?

Если $\Box$ — модальность времени, «всегда», то $\Box\varphi \to \varphi$ — это закон модальной логики.

Если студенты всегда ходят на лекции, то они ходят на лекции.

А вот если $\Box$ — деонтическая модальность, «должны», то формула $\Box\varphi \to \varphi$ уже не может претендовать на статус логического закона.

Если студенты должны ходить на лекции, то они ходят на лекции.

Это неправда, а законы логики не зависят от прихоти студентов.

МОДАЛЬНЫЕ ЛОГИКИ

Семантика Крипке модальных формул

Определим самое общее отношение выполнимости для модальных формул.

Пусть $\mathcal{P} = \{P_1, P_2, \ldots, P_n, \ldots\}$ — множество атомарных формул (элементарные высказывания).

Модальная интерпретация или модель Крипке — это реляционная система $I = \langle W, \mathbf{R}, \xi \rangle$, в которой

1. $W \neq \emptyset$ — множество состояний (возможные миры);
2. $\mathbf{R} \subseteq S \times S$ — отношение достижимости на W,
3. $\xi : W \times \mathcal{P} \to \{\mathbf{true}, \mathbf{false}\}$ — оценка атомарных формул.

Система $\langle W, \mathbf{R} \rangle$ называется шкалой Крипке (frame).

Если $(w, w') \in \mathbf{R}$, то возможный мир w' называется альтернативным миром для s.

МОДАЛЬНЫЕ ЛОГИКИ

Отношение выполнимости для модальных формул

Пусть $I = \langle W, \mathbf{R}, \xi \rangle$ — модель Крипке. Тогда отношение выполнимости $I, s \models \varphi$ формулы φ в мире s модели I определяется так:

1. если $\varphi = P \in \mathcal{P}$, то $I, s \models \varphi \iff \xi(w, P) = \mathbf{true}$;
2. $I, w \models \varphi_1 \& \varphi_2 \iff I, w \models \varphi_1$ и $I, w \models \varphi_2$;
3. $I, w \models \varphi_1 \vee \varphi_2 \iff I, w \models \varphi_1$ или $I, w \models \varphi_2$;
4. $I, w \models \varphi_1 \to \varphi_2 \iff I, w \not\models \varphi_1$ или $I, w \models \varphi_2$;
5. $I, w \models \neg\varphi_1 \iff I, w \not\models \varphi_1$;
6. $I, w \models \Box\varphi \iff$
   для любого альтернативного мира w'
   если $\langle w, w' \rangle \in \mathbf{R}$, то $I, w' \models \varphi$;
7. $I, w \models \Diamond\varphi \iff$
   существует такой альтернативный мир w',
   что $\langle w, w' \rangle \in \mathbf{R}$ и $I, w' \models \varphi$.

МОДАЛЬНЫЕ ЛОГИКИ

Пример

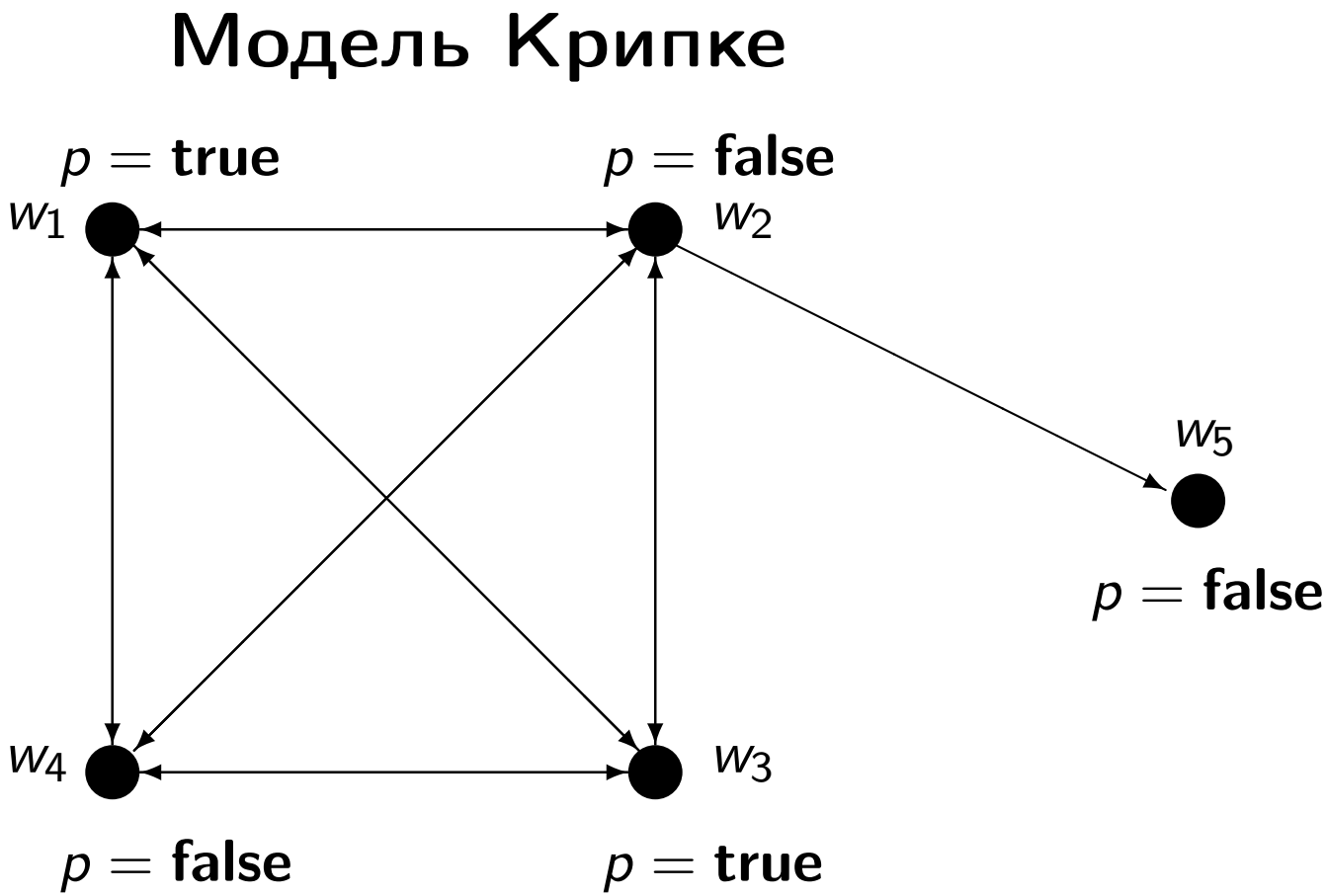
$I, w_1 \models \Diamond p$

$I, w_1 \not\models \Box p$

$I, w_1 \models \Box\Diamond p$

$I, w_5 \models \Box p$

$I, w_5 \not\models \Diamond p$



МОДАЛЬНЫЕ ЛОГИКИ

Простейшие свойства

1. $\models \Diamond\varphi \equiv \neg\Box\neg\varphi$;
2. $\models \Box(\varphi_1 \to \varphi_2) \to (\Box\varphi_1 \to \Box\varphi_2)$;
3. $\models \varphi \Longrightarrow \models \Box\varphi$ (правило необходимости).

В разных приложениях модальность необходимого может пониматься по разному. Отсюда большое разнообразие модальных логик. В разных модальных логиках отношение выполнимости определяется на разных классах шкал. Каждая разновидность шкал (отношения достижимости $\mathbf{R}$) характеризуется определенным законом (формулой) модальной логики.

МОДАЛЬНЫЕ ЛОГИКИ

Характеристические формулы

1. $\Box\varphi \to \varphi$

   рефлексивные шкалы

   $\forall w \mathbf{R}(w, w)$;
2. $\Box\varphi \to \Box\Box\varphi$

   транзитивные шкалы

   $\forall w_1 \forall w_2 \forall w_3 (\mathbf{R}(w_1, w_2) \& \mathbf{R}(w_2, w_3) \to \mathbf{R}(w_1, w_3))$;
3. $\Diamond\Box\varphi \to \Box\varphi$

   симметричные шкалы

   $\forall w_1 \forall w_2 (\mathbf{R}(w_1, w_2) \to \mathbf{R}(w_2, w_1))$.

Рассмотрим некоторые разновидности модальных логик, которые используются информатике.

МОДАЛЬНЫЕ ЛОГИКИ

Эпистемические логики и мультагентные системы

Эпистемические логики — это разновидности модальных логик, изучающие модальности знания и мнения (веры) идеализированных агентов. Интерес представляют вопросы о том, какими знаниям располагает субъект, насколько он осознает свои знания (и незнания), и какие причинно-следственные связи возникают между утверждениями, касающимися вопросов знания и веры.

В эпистемической логике модальный оператор $\Box\varphi$ следует прочитывать «Я знаю, что φ», а $\Diamond\varphi$ — «Я допускаю, что φ».

МОДАЛЬНЫЕ ЛОГИКИ

Эпистемические логики и мультагентные системы

Основные законы (аксимы) эпистемической логики:

1. Аксиома адекватности знания: $\Box\varphi \rightarrow \varphi$
   «Мои знания верны».
2. Аксиома позитивной интроспекции: $\Box\varphi \rightarrow \Box\Box\varphi$
   «Я вполне представляю все, что мне известно».
3. Аксиома негативной интроспекции: $\Diamond\Box\varphi \rightarrow \Box\varphi$
   «Я вполне сознаю, что именно мне неизвестно».

Но чаще всего возникают задачи, когда коллектив субъектов (мультиагентная система) пытается совместными усилиями или в конкурентной борьбе достичь какой-то цели. В таком случае каждый агент должен принимать в расчет не только знания о предметной области, но и представления о том, какими знаниями располагают другие агенты.

МОДАЛЬНЫЕ ЛОГИКИ

Задача.

Три мудреца спорили о том, кто из них мудрее. Прохожий взялся разрешить их спор. Он сказал: «У меня в мешке пять шапок: 3 черных и 2 белых. Я завяжу вам глаза, надену каждому на голову одну из шапок, а потом развяжу глаза. Тот из вас, кто первым догадается, какого цвета шапка у него на голове, будет признан мудрейшмим». Мудрецы согласились, и прохожий исполнил все то, о чем он говорил. После того, как с глаз мудрецов были сняты повязки, некоторое время никто не произнес ни слова. И после этого один из мудрецов заявил: «На моей голове черная шапка». Он оказался прав, и был признан мудрейшим.

Вопрос: Докажите, что мудрейший из мудрых слеп.

МОДАЛЬНЫЕ ЛОГИКИ

Эпистемические логики и мультагентные системы

В мультиагентных системах нужно ввести более специальные модальные операторы.

Пусть $\mathcal{A} = \{a_1, a_2, \ldots a_n\}$ — множество агентов. Тогда

$\Box_a \varphi$ означает «Агент a знает, что φ верно».

$\Box_C \varphi$ означает «Все агенты знают, что φ верно».

Специальные разновидности эпистемических логик применяются для описания и проверки требований безопасности сетевых протоколов.

МОДАЛЬНЫЕ ЛОГИКИ

Темпоральные логики

Темпоральные (временные) логики применяются для описания и исследования причинно-следственных зависимостей, развивающихся во времени.
Модальный оператор $\Box$ означает «всегда»,
а оператор $\Diamond$ — «когда-нибудь».

Семантика темпоральных логик существенно зависит от той математической модели, которая используется для описания феномена времени. В самом общем случае в качестве модели времени можно взять любое частично упорядоченное множество. Элементы этого множества соответствуют различным моментам времени.

В качестве темпоральных моделей могут выступать любые модели Крипке, построенные на основе частично упорядоченных шкал. Разные отношения частичного порядка порождают разные темпоральные логики.

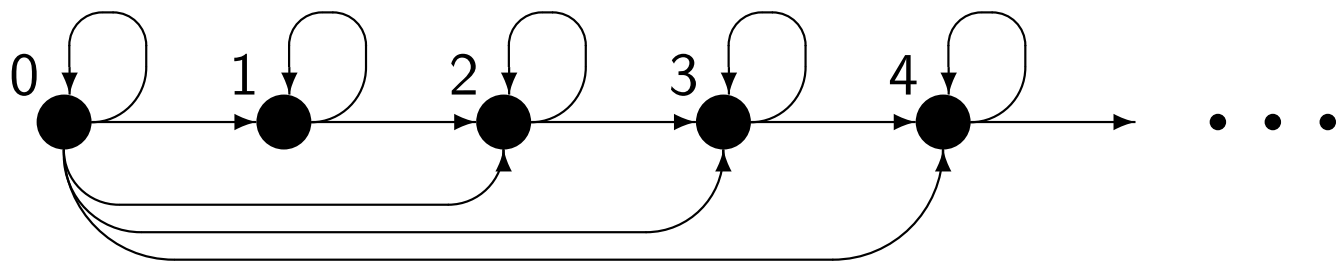
МОДАЛЬНЫЕ ЛОГИКИ

Темпоральные логики

Поскольку вычисление — это процесс, развивающийся во времени, состояния которого находятся в причинно-следственной связи друг с другом, темпоральные логики используются для спецификации и верификации программ. Наиболее широкое распространение получили две разновидности темпоральных логик.

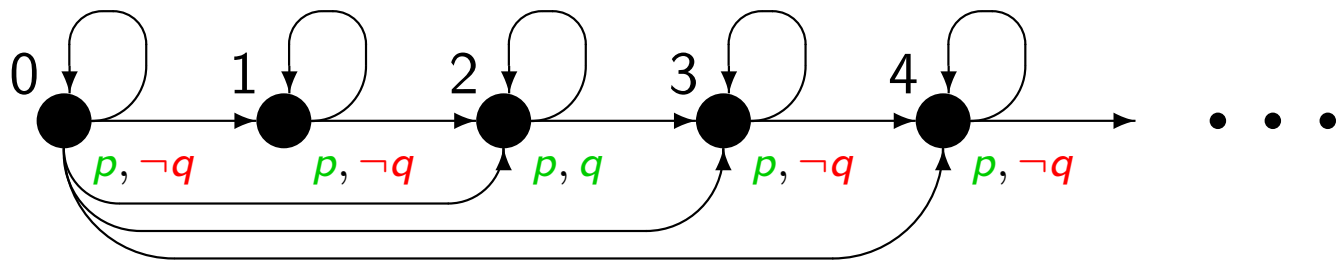
Логика линейного времени LTL

Шкала Крипке для LTL (L inear T emporal L ogics) — это натуральный ряд с естественным отношением порядка $\langle \mathbb{N}, \leq \rangle$.



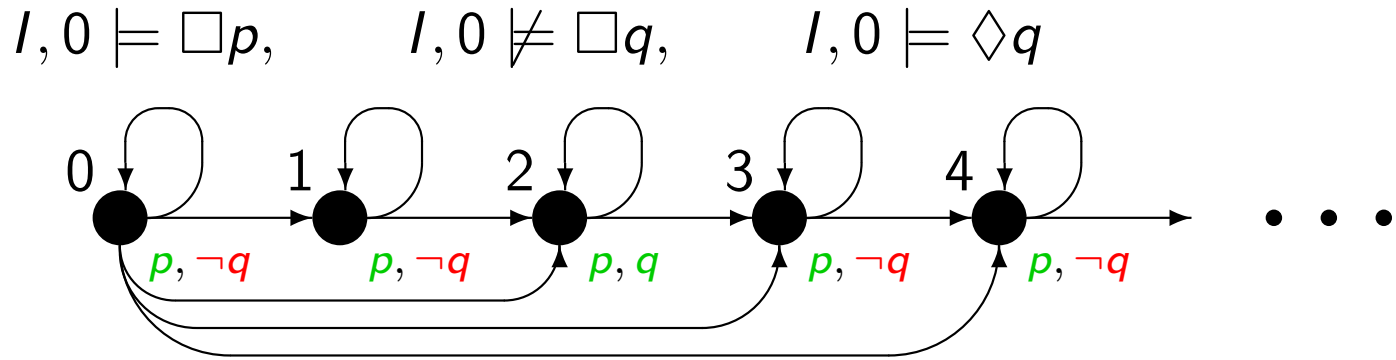
МОДАЛЬНЫЕ ЛОГИКИ

Логика линейного времени LTL



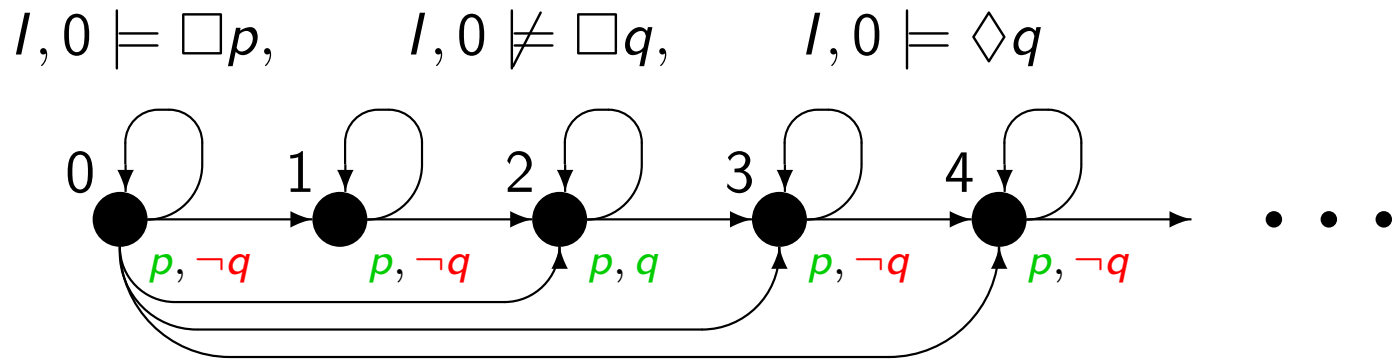
МОДАЛЬНЫЕ ЛОГИКИ

Логика линейного времени LTL



МОДАЛЬНЫЕ ЛОГИКИ

Логика линейного времени LTL

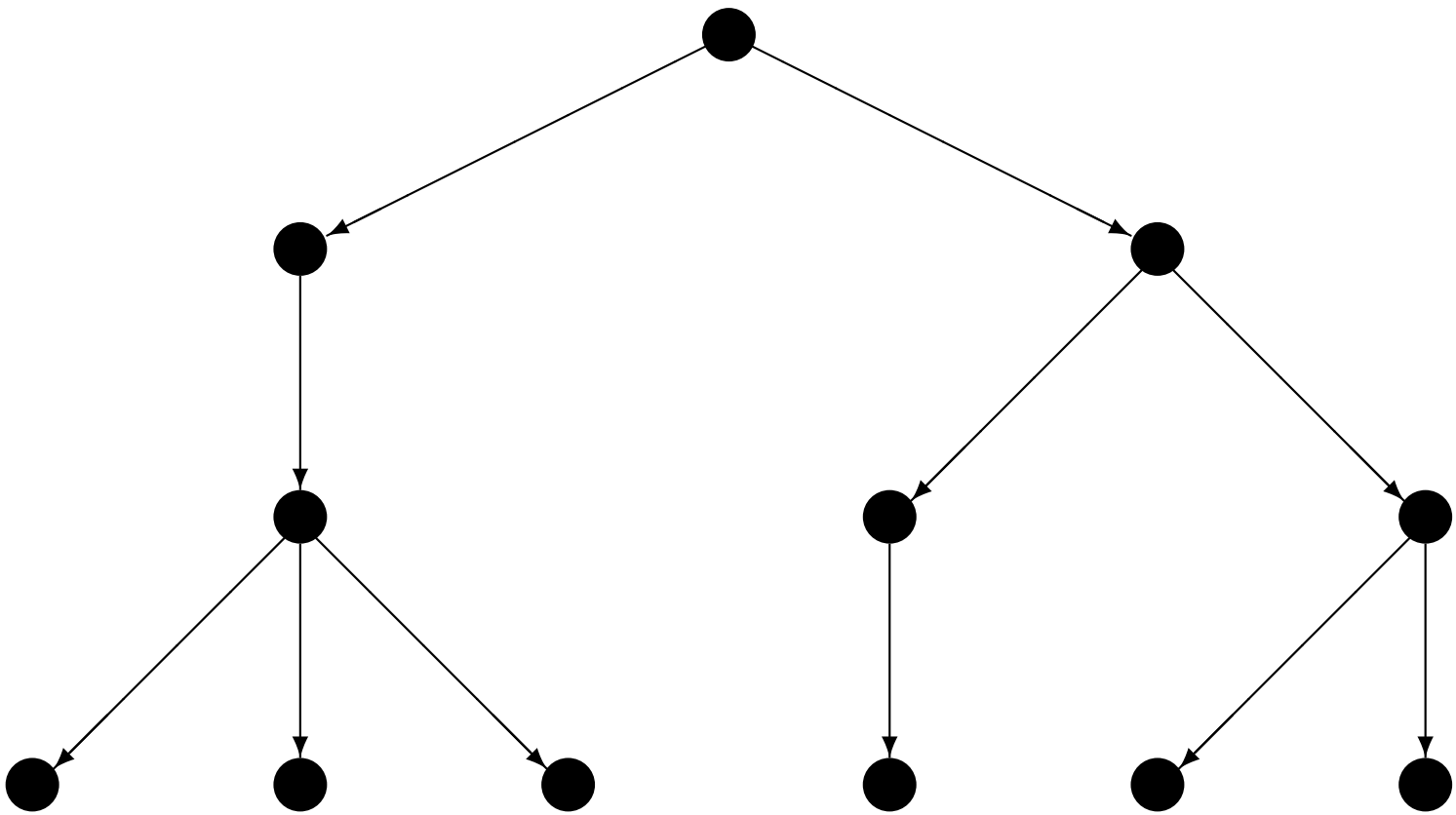


Применение LTL для верификации моделей программ более подробно будет обсуждаться в последующих лекциях.

МОДАЛЬНЫЕ ЛОГИКИ

Темпоральные логики

В других темпоральных логиках время — это ветвящаяся структура; в каждый момент времени может быть несколько альтернатив дальнейшего развития событий.



МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

Темпоральные логики такого вида называются логиками ветвящегося времени (BTL, Branching Time Logics).

Одной из логик ветвящегося времени является логика деревьев вычислений (CTL, Computational Tree Logic), используемая для спецификации и верификации распределенных программ и микроэлектронных схем.

В логике CTL имеются темпоральные операторы двух типов — универсальные и экзистенциальные.

$$\forall\Box, \quad \forall\Diamond, \qquad\qquad \exists\Box, \quad \exists\Diamond.$$

Тип темпорального оператора указывет на то, будет ли выполнимость формулы проверяться на всех ветвях древесной модели или только на одной ветви.

МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

Пусть $I = \langle S, \mathbf{R}, \xi \rangle$ — древесная модель Крипке для логики CTL, $s_0 \in S$ — одно из состояний модели. Тогда

$I, s_0 \models \forall\Box\varphi \iff$
в каждом состоянии s, достижимом из состояния s_0, верно $I, s \models \varphi$;

$I, s_0 \models \exists\Box\varphi \iff$
существует ветвь, исходящая из состояния s_0, в каждом состоянии s которой верно $I, s \models \varphi$;

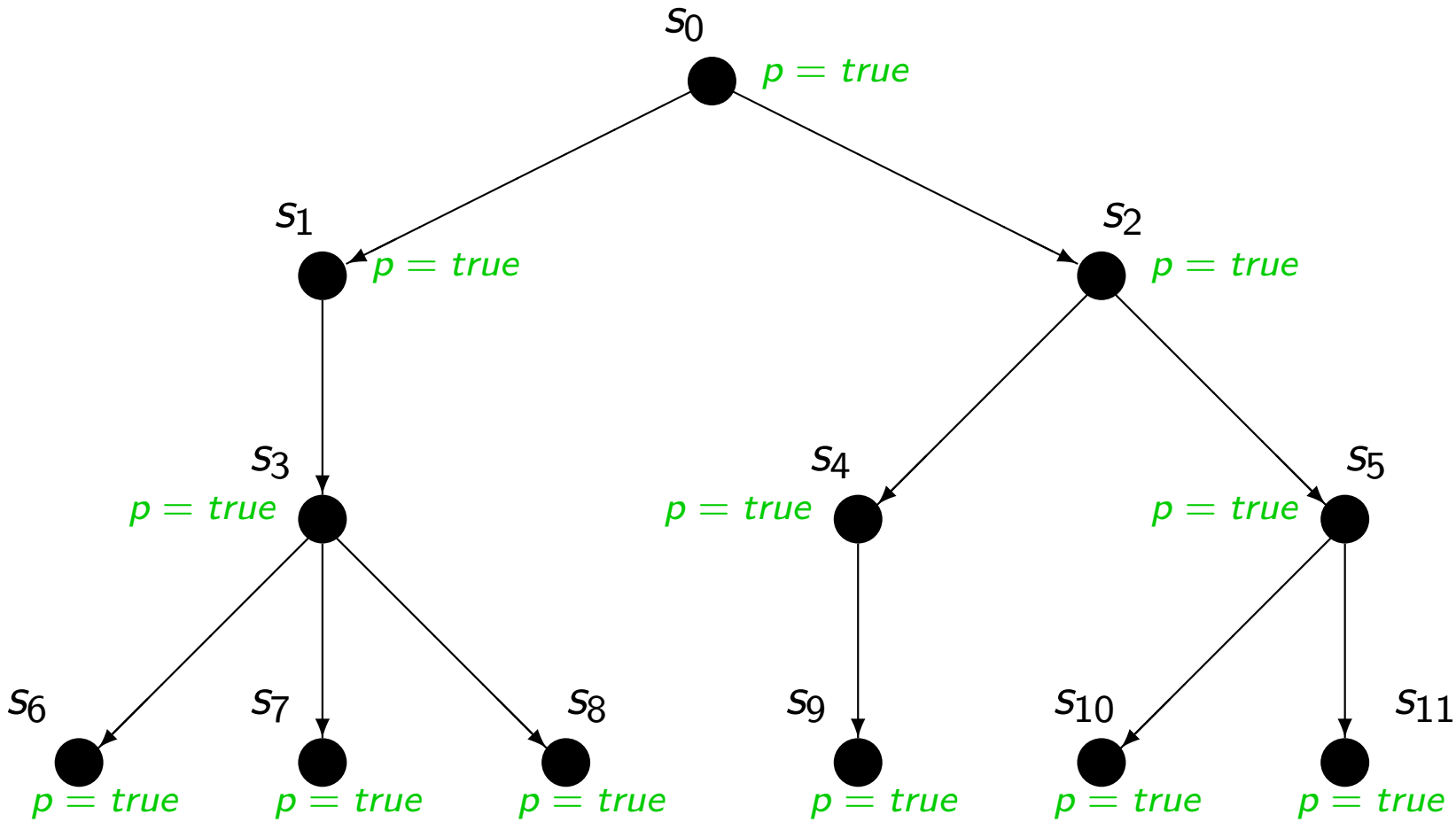
$I, s_0 \models \forall\Diamond\varphi \iff$
в каждой ветви, исходящей из состояния s_0, есть состояние s, в котором верно $I, s \models \varphi$;

$I, s_0 \models \exists\Diamond\varphi \iff$
существует ветвь, исходящая из состояния s_0, в одном из состоянии s которой верно $I, s \models \varphi$.

МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

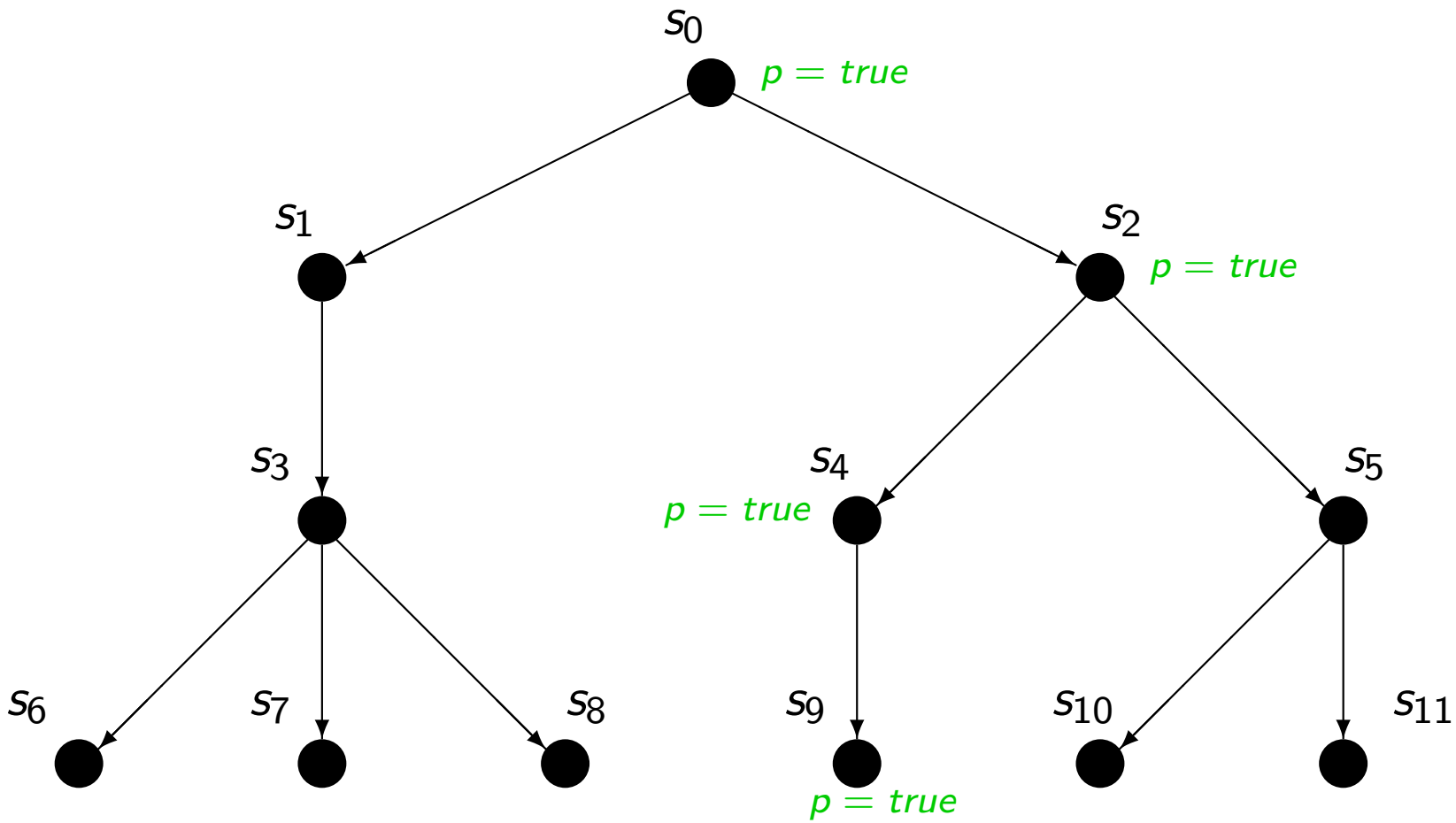
$$I, s_0 \models \forall\Box p$$



МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

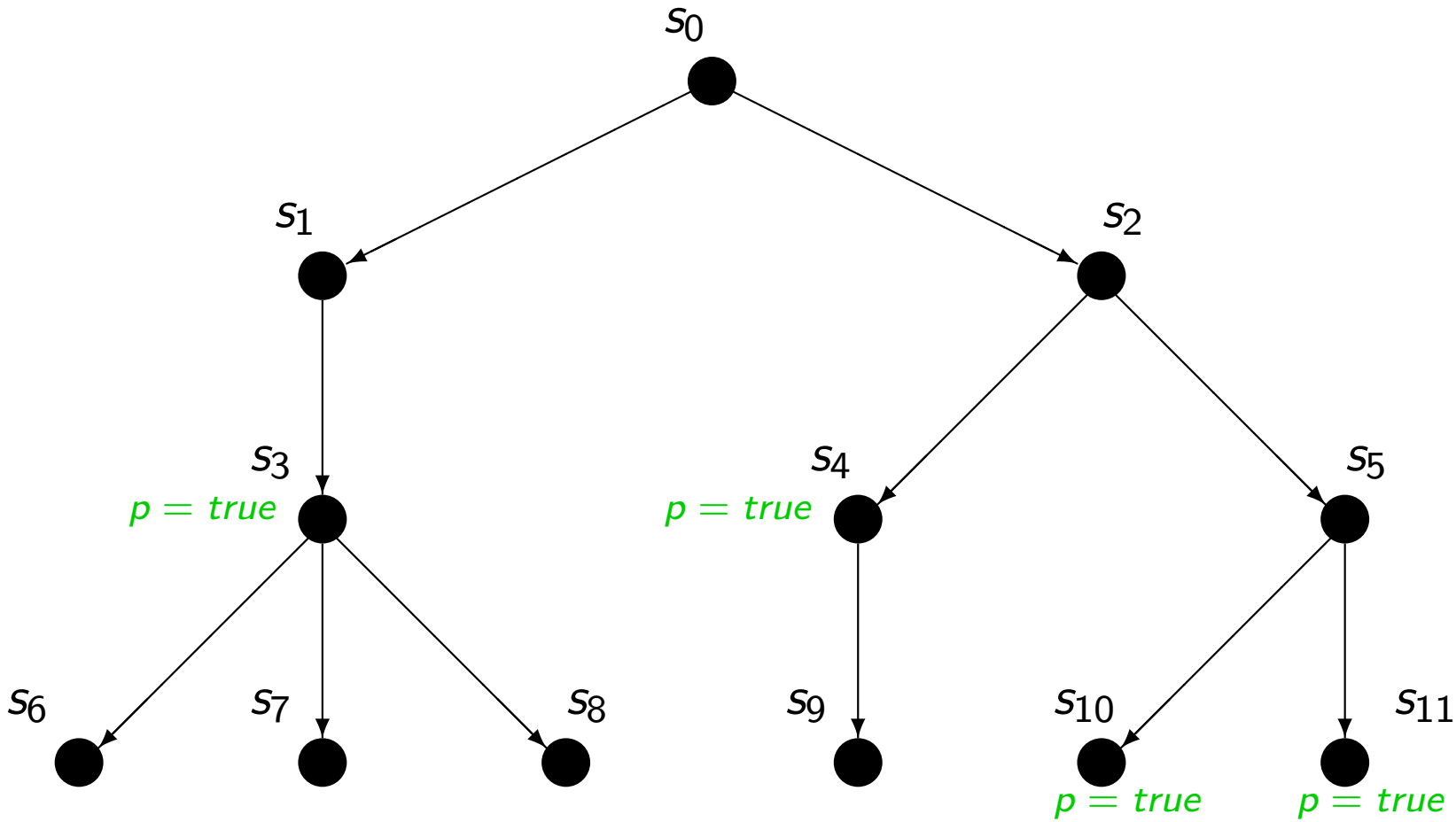
$$I, s_0 \models \exists\Box p$$



МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

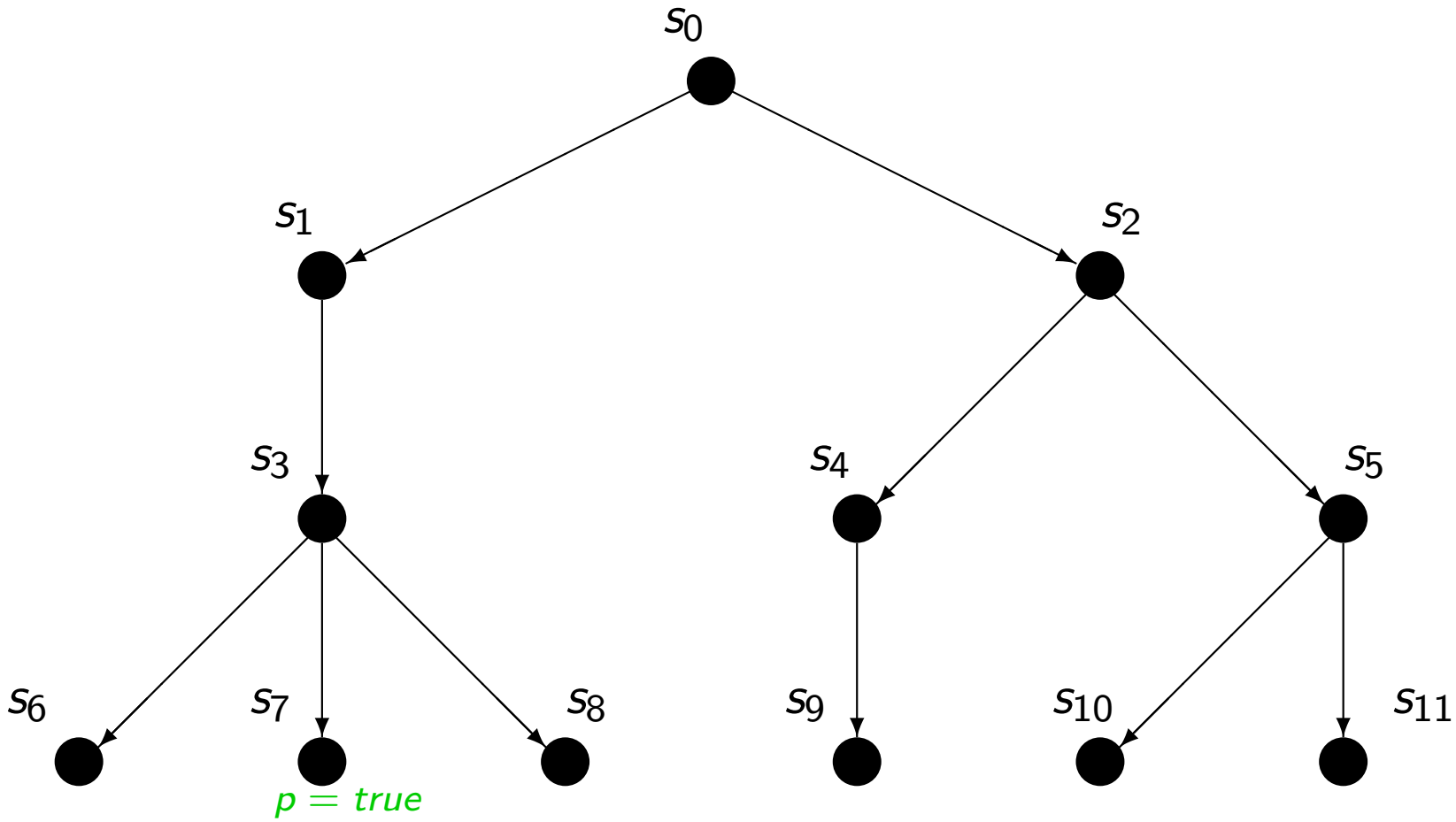
$$I, s_0 \models \forall \Diamond p$$



МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

$$I, s_0 \models \exists \Diamond p$$



МОДАЛЬНЫЕ ЛОГИКИ

Логика деревьев вычислений CTL

Формулы CTL можно использовать для формальной спецификации многих интересных свойств поведения программ

$\forall\Box\ \exists\Diamond\ \mathit{Restart}$:
на любом этапе функционирования системы можно осуществить ее перезапуск;

$\forall\Box\ (\mathit{Request}\ \rightarrow\ \forall\Diamond\ \mathit{Response})$:
когда бы ни был послан запрос, рано или поздно на него обязательно поступит отклик.

МОДАЛЬНЫЕ ЛОГИКИ

А как проверить,
что вычисления программ
удовлетворяют заданным спецификациям?

И можно ли эту проверку
автоматизировать?

КОНЕЦ ЛЕКЦИИ 18-19.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 20.

Правильные программы. Императивные программы. Задача верификации программ. Логика Хоара. Автоматическая проверка правильности программ.

ПРАВИЛЬНЫЕ ПРОГРАММЫ

Какая компьютерная программа считается хорошей?

Та, которая работает ПРАВИЛЬНО и **эффективно** .

А какая программа считается правильной?

Правильной считается та программа, которая выполняет в точности то, что от нее требуется.

А как убедиться, что программа выполняет то, что от нее требуется?

Для этого нужно

1. Описать строго (формально) требования правильности вычислений;
2. Проверить, что все вычисления программы удовлетворяют этим требованиям.

ПРАВИЛЬНЫЕ ПРОГРАММЫ

Описание требований правильности функционирования программы называется спецификацией программы.

Проверка соблюдения вычислениями программы требований правильности функционирования называется верификацией программы.

Если спецификации программ записать на формальном логическом языке и строго определить операционную семантику программ, то для доказательства правильности программ можно использовать методы математической логики (логический вывод).

ПРАВИЛЬНЫЕ ПРОГРАММЫ

Формальная верификация программ

Преимущества	Проблемы
1. Абсолютно точная проверка правильности программ.	1. Как заставить программистов писать формальные спецификации?
2. Возможность автоматизации построения логического вывода.	2. Как заставить прувер работать эффективно?

И, тем не менее, попробуем...

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Определим синтаксис и семантику императивных программ.

Пусть задана сигнатура $\sigma = \langle Const, Func, Pred \rangle$, в которой определно множество термов $Term$ и множество атомарных формул $Atom$.
Условимся, что $\Leftarrow$ — это служебный символ, не принадлежащий сигнатуре σ.

Определение

присваивание ::= «переменная» $\Leftarrow$ «терм»
условие ::= «атом» | ($\neg$условие) |
(условие & условие) | (условие $\vee$ условие)
программа ::= присваивание |
программа ; программа |
if «условие» **then** программа **else** программа **fi** |
while условие **do** программа **od**

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Пример

Программа вычисления наибольшего общего делителя двух натуральных чисел.

$Const = \{0, 1, 2, \dots\}$,
$Func = \{+^{(2)}, -^{(2)}\}$,
$Pred = \{=^{(2)}, >^{(2)}, <^{(2)}\}$

while $\neg(x = y)$
do
if $x > y$
then $x \Leftarrow x - y$
else $y \Leftarrow y - x$
fi
od

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Операционная семантика императивных программ

Семантика задает смысл (значение) синтаксических конструкций (слов, формул, программ и пр.).

Значением императивной программы является **отношение вход–выход** между входными данными и результатом вычисления.

Отношение вход–выход программы определяется при помощи отношения переходов между **состояниями вычисления** программы.

Состояние вычисления программы определяется двумя компонентами — **состоянием управления** и **состоянием данных** .

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Определение (состояния вычисления)

Пусть **Var** — это множество переменных, а *GTerm* — это множество основных термов сигнатуры σ.

Оценкой переменных (состоянием данных) будем называть всякое отображение (подстановку) $\theta : \mathbf{Var} \to \mathit{GTerm}$.

Состоянием управления будем называть всякую программу, а также специальный символ $\varnothing$.

Состоянием вычисления будем называть всякую пару $\langle \pi, \theta \rangle$, где π — состояние управления, а θ — оценка переменных.

Запись State_σ будет обозначать множество всевозможных состояний вычислений сигнатуры σ.

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Определение (отношения переходов)

Пусть I — это интерпретация сигнатуры σ.

Тогда отношение переходов для императивных программ — это бинарное отношение $\longrightarrow_I$ на множестве состояний вычисления $State_\sigma$, удовлетворяющее следующим требованиям:

ASS: $\langle x \Rightarrow t,\ \theta \rangle \longrightarrow_I \langle \varnothing,\ \{x/t\}\theta \rangle$;

COMP_$\varnothing$: $\langle \pi_1; \pi_2,\ \theta \rangle \longrightarrow_I \langle \pi_2,\ \eta \rangle$
тогда и только тогда, когда $\langle \pi_1,\ \theta \rangle \longrightarrow_I \langle \varnothing,\ \eta \rangle$;

COMP: $\langle \pi_1; \pi_2,\ \theta \rangle \longrightarrow_I \langle \pi_1'; \pi_2,\ \eta \rangle$
тогда и только тогда, когда $\langle \pi_1,\ \theta \rangle \longrightarrow_I \langle \pi_1',\ \eta \rangle$ и $\pi_1' \neq \varnothing$;

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Определение (отношения переходов)

IF_1: $\langle \textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}, \ \theta \rangle \ \longrightarrow_I \ \langle \pi_1, \ \theta \rangle$
тогда и только тогда, когда $I \models C\theta$;

IF_0: $\langle \textbf{if } C \textbf{ then } \pi_1 \textbf{ else } \pi_2 \textbf{ fi}, \ \theta \rangle \ \longrightarrow_I \ \langle \pi_2, \ \theta \rangle$
тогда и только тогда, когда $I \not\models C\theta$;

WHILE_1: $\langle \textbf{while } C \textbf{ do } \pi \textbf{ od}, \ \theta \rangle \ \longrightarrow_I \ \langle \pi; \textbf{ while } C \textbf{ do } \pi \textbf{ od}, \ \theta \rangle$
тогда и только тогда, когда $I \models C\theta$;

WHILE_0: $\langle \textbf{while } C \textbf{ do } \pi \textbf{ od}, \ \theta \rangle \ \longrightarrow_I \ \langle \varnothing, \ \theta \rangle$
тогда и только тогда, когда $I \not\models C\theta$.

Отношение переходов $\longrightarrow_I$ определяет, как изменяется состояние вычисления за один шаг работы интерпретатора императивных программ.

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Определение (вычисления программы)

Пусть π_0 — это императивная программа, θ_0 — оценка переменных.

Частичным вычислением программы π_0 на оценке переменных θ_0 в интерпретации I называется последовательность (конечная или бесконечная) состояний вычисления

$$\langle \pi_0, \theta_0 \rangle, \langle \pi_1, \theta_1 \rangle, \ldots, \langle \pi_{n-1}, \theta_{n-1} \rangle, \langle \pi_n, \theta_n \rangle, \ldots,$$

в которой для любого n, $n \geq 1$, выполняется отношение $\langle \pi_{n-1}, \theta_{n-1} \rangle \longrightarrow_I \langle \pi_n, \theta_n \rangle$.

Вычислением программы π_0 на оценке переменных θ_0 в интерпретации I называется всякое частичное вычисление, которое нельзя продолжить.

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Пример

Пусть I — интерпретация сигнатуры $\sigma = \langle Const, Func, Pred \rangle$:
$Const = \{0, 1, 2, \ldots, \}$, $Func = \{+^{(2)}, -^{(2)}\}$,
$Pred = \{=^{(2)}, >^{(2)}, <^{(2)}\}$,
предметной областью которой является множество натуральных чисел $\mathcal{N}_0$ с обычными арифметическими операциями и отношениями.

Рассмотрим вычисление программы

π_0: **while** $\neg(x = y)$
do if $x > y$ **then** $x \Leftarrow x - y$ **else** $y \Leftarrow y - x$ **fi od**

на оценке переменных $\theta_0 = \{x/4, y/6\}$.

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

π_0: **while** $\neg(x = y)$
do if $x > y$ **then** $x \Leftarrow x - y$ **else** $y \Leftarrow y - x$ **fi od**

$\theta_0 = \{x/4, y/6\}$

Пример

$$\langle \pi_0,\ \{x/4, y/6\} \rangle$$
$$\downarrow_I$$
$$\langle \textbf{if } x > y \textbf{ then } x \Leftarrow x - y \textbf{ else } y \Leftarrow y - x \textbf{ fi } ; \pi_0,\ \{x/4, y/6\} \rangle$$
$$\downarrow_I$$
$$\langle y \Leftarrow y - x; \pi_0,\ \{x/4, y/6\} \rangle$$
$$\downarrow_I$$
$$\langle \pi_0,\ \{x/4, y/6 - 4\} \rangle$$

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

π_0: **while** $\neg(x = y)$
do if $x > y$ **then** $x \Leftarrow x - y$ **else** $y \Leftarrow y - x$ **fi od**

$\theta_0 = \{x/4, y/6\}$

Пример

$$\langle \pi_0,\ \{x/4, y/6-4\}\rangle$$
$$\downarrow_I$$
$$\langle \textbf{if } x > y \textbf{ then } x \Leftarrow x - y \textbf{ else } y \Leftarrow y - x \textbf{ fi } ; \pi_0,\ \{x/4, y/6-4\}\rangle$$
$$\downarrow_I$$
$$\langle x \Leftarrow x - y; \pi_0,\ \{x/4, y/6-4\}\rangle$$
$$\downarrow_I$$
$$\langle \pi_0,\ \{x/4-(6-4), y/6-4\}\rangle$$
$$\downarrow_I$$
$$\langle \varnothing,\ \{x/4-(6-4), y/6-4\}\rangle$$

ИМПЕРАТИВНЫЕ ПРОГРАММЫ

Как следует из определения, любое вычисление либо является бесконечной последовательностью, либо завершается состоянием $\langle \varnothing, \eta \rangle$. В последнем случае оценка η называется результатом вычисления.

Будем использовать запись $\longrightarrow_I^*$ для обозначения рефлексивного и транзитивного замыкания отношения переходов $\longrightarrow_I$.

Тогда оценка переменных η является результатом вычисления программы π на оценке переменных θ в интерпретации I в том и только том случае, когда выполняется отношение

$$\langle \pi, \theta \rangle \longrightarrow_I^* \langle \varnothing, \eta \rangle.$$

ЗАДАЧА ВЕРИФИКАЦИИ ПРОГРАММ

Неформальная постановка.

Программа π считается (частично) корректной , если для любых начальных данных, удовлетворяющих определенному условию φ, результат вычисления (если вычисление завершается) удовлетворяет определенному условию ψ.

Ограничение φ, которое налагается на начальные данные, называется предусловием , а требование ψ, которому должны удовлетворять результаты вычисления, называется постусловием программы.

Задача верификации программы π заключается в проверке частичной корректности программы π относительно заданного предусловия φ и заданного постусловия ψ.

ЗАДАЧА ВЕРИФИКАЦИИ ПРОГРАММ

Формальная постановка.

Расширим множество формул логики предикатов, введя в рассмотрение в качестве формул выражения нового специального вида.

Определение

Триплетом Хоара (тройкой Хоара) называется всякое выражение вида

$$\varphi\{\pi\}\psi,$$

где φ, ψ — формулы логики предикатов,
а π — императивная программа.

Обозначим HT_σ множество триплетов Хоара сигнатуры σ.

ЗАДАЧА ВЕРИФИКАЦИИ ПРОГРАММ

Выполнимость триплетов Хоара в интерпретациях определяется так:

$$I \models \varphi\{\pi\}\psi \quad \Longleftrightarrow \quad \begin{array}{l} \text{для любых оценок переменных } \theta, \eta, \\ \text{если } I \models \varphi\theta \text{ и } \langle \pi, \theta \rangle \longrightarrow_I^* \langle \varnothing, \eta \rangle, \\ \text{то } I \models \psi\eta. \end{array}$$

Определение (частичной корректности программы)

Пусть φ, ψ — формулы логики предикатов, а π — императивная программа.

Программа π называется частично корректной в интерпретации I относительно предусловия φ и постусловия ψ, если триплет $\varphi\{\pi\}\psi$ выполним в интерпретации I, т. е.

$$I \models \varphi\{\pi\}\psi \ .$$

ЛОГИКА ХОАРА

Как же доказать частичную корректность программы?

Попробуем построить систему правил вывода, аналогичных правилам вывода для семантических таблиц.

Такую систему предложил в 1968 г. Хоар. Правила вывода Хоара имеют вид

$$\frac{\Phi}{\Psi}, \quad \frac{\Phi}{\varphi}, \quad \frac{\Phi}{\Psi_1, \Psi_2}, \frac{\Phi}{\varphi, \Psi, \psi},$$

где $\Phi, \Psi, \Psi_1, \Psi_2$ — триплеты Хоара,
φ, ψ — формулы логики предикатов.

ЛОГИКА ХОАРА

Правила вывода Хоара

ASS: $$\frac{\varphi\{x/t\}\ \{x \Leftarrow t\}\ \varphi}{\textbf{true}},$$

CONS: $$\frac{\varphi\{\pi\}\psi}{\varphi \to \varphi',\ \varphi'\{\pi\}\psi',\ \psi' \to \psi},$$

COMP: $$\frac{\varphi\{\pi_1;\ \pi_2\}\psi}{\varphi\{\pi_1\}\chi,\ \chi\{\pi_2\}\psi},$$

IF: $$\frac{\varphi\ \{\textbf{if}\ C\ \textbf{then}\ \pi_1\ \textbf{else}\ \pi_2\ \textbf{fi}\}\ \psi}{(\varphi\&C)\ \{\pi_1\}\ \psi,\ (\varphi\&\neg C)\ \{\pi_2\}\ \psi},$$

WHILE: $$\frac{\varphi\ \{\textbf{while}\ C\ \textbf{do}\ \pi\ \textbf{od}\}\ (\varphi\&\neg C)}{(\varphi\&C)\ \{\pi\}\ \varphi}.$$

ЛОГИКА ХОАРА

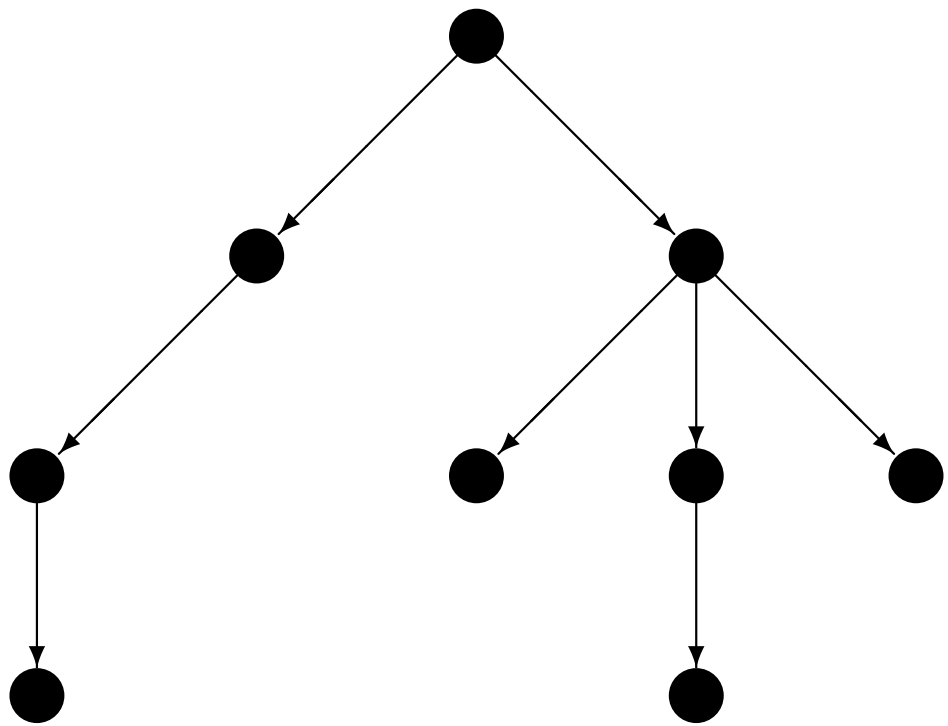
Определение вывода в логике Хоара

Вывод в логике Хоара триплета $\Phi_0 = \varphi_0 \ \{\pi_0\} \ \psi_0$ — это корневое дерево,

ЛОГИКА ХОАРА

Определение вывода в логике Хоара

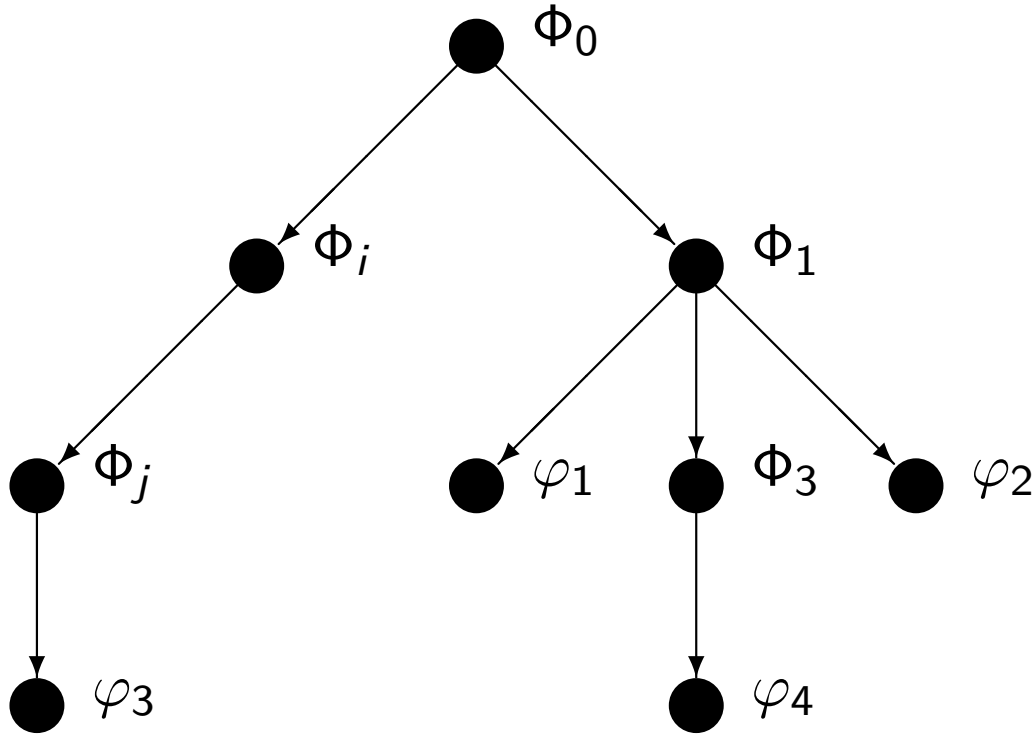
Вывод в логике Хоара триплета $\Phi_0 = \varphi_0 \ \{\pi_0\} \ \psi_0$ — это корневое дерево,



ЛОГИКА ХОАРА

Определение вывода в логике Хоара

Вывод в логике Хоара триплета $\Phi_0 = \varphi_0 \{\pi_0\} \psi_0$ — это корневое дерево, вершинами которого служат триплеты и формулы логики предикатов и при этом

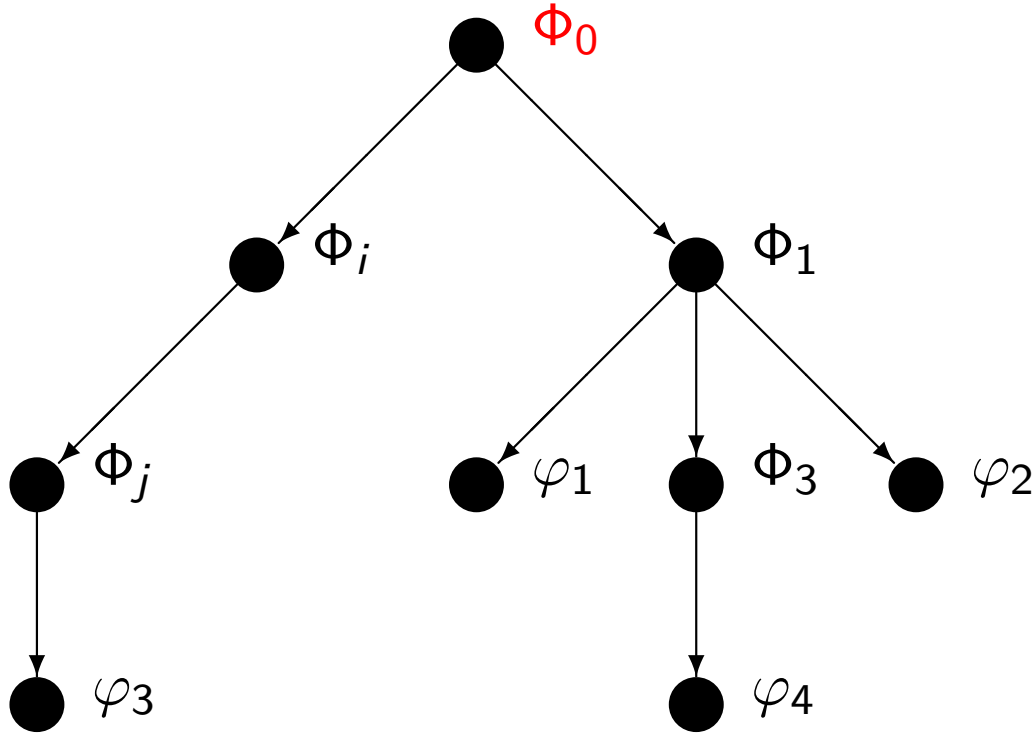


ЛОГИКА ХОАРА

Определение вывода в логике Хоара

Вывод в логике Хоара триплета $\Phi_0 = \varphi_0 \{\pi_0\} \psi_0$ — это корневое дерево, вершинами которого служат триплеты и формулы логики предикатов и при этом

1) корнем дерева является триплет Φ_0;



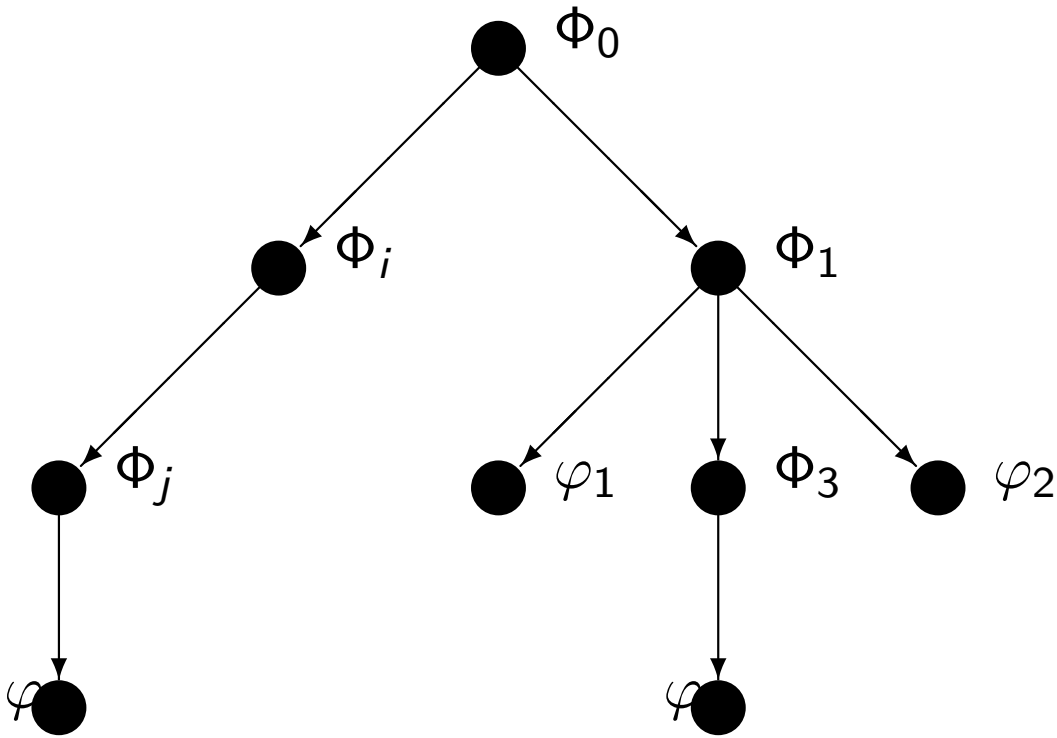
ЛОГИКА ХОАРА

Определение вывода в логике Хоара

2) из вершины Φ_i исходят дуги в вершину Φ_j

$\Longleftrightarrow$

$\frac{\Phi_i}{\Phi_j}$ — правило табличного вывода;



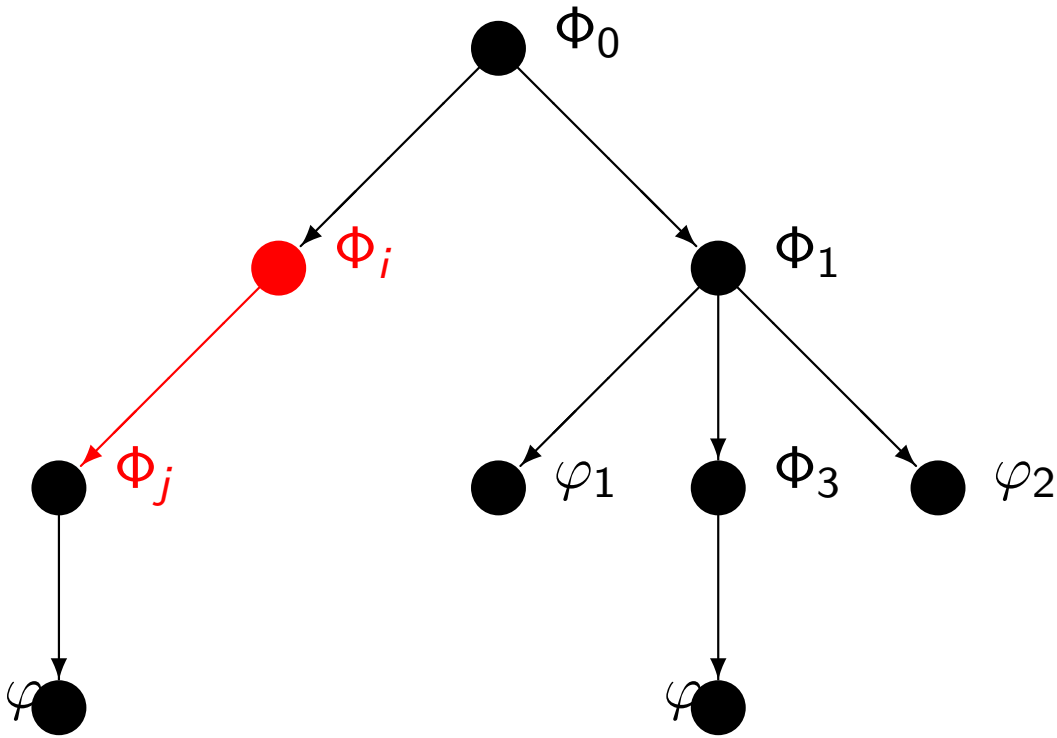
ЛОГИКА ХОАРА

Определение вывода в логике Хоара

2) из вершины Φ_i исходят дуги в вершину Φ_j

$\Longleftrightarrow$

$\frac{\Phi_i}{\Phi_j}$ — правило табличного вывода;



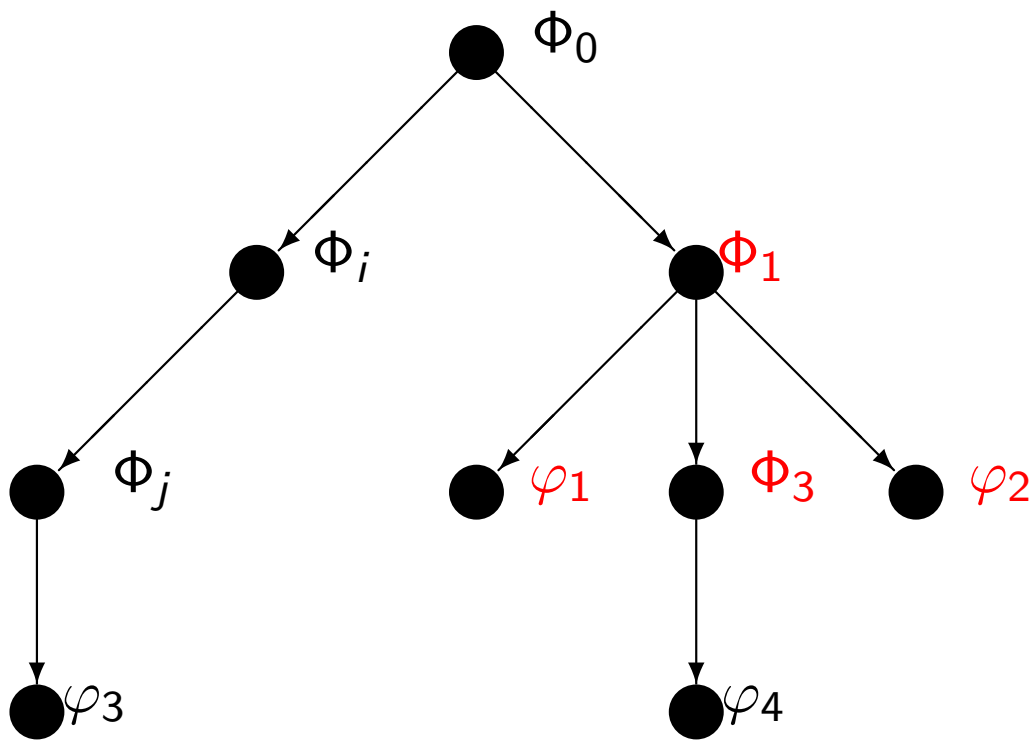
ЛОГИКА ХОАРА

Определение вывода в логике Хоара

2) из вершины Φ_1 исходят дуги в вершины φ_1, Φ_3, φ_2

$\Longleftrightarrow$

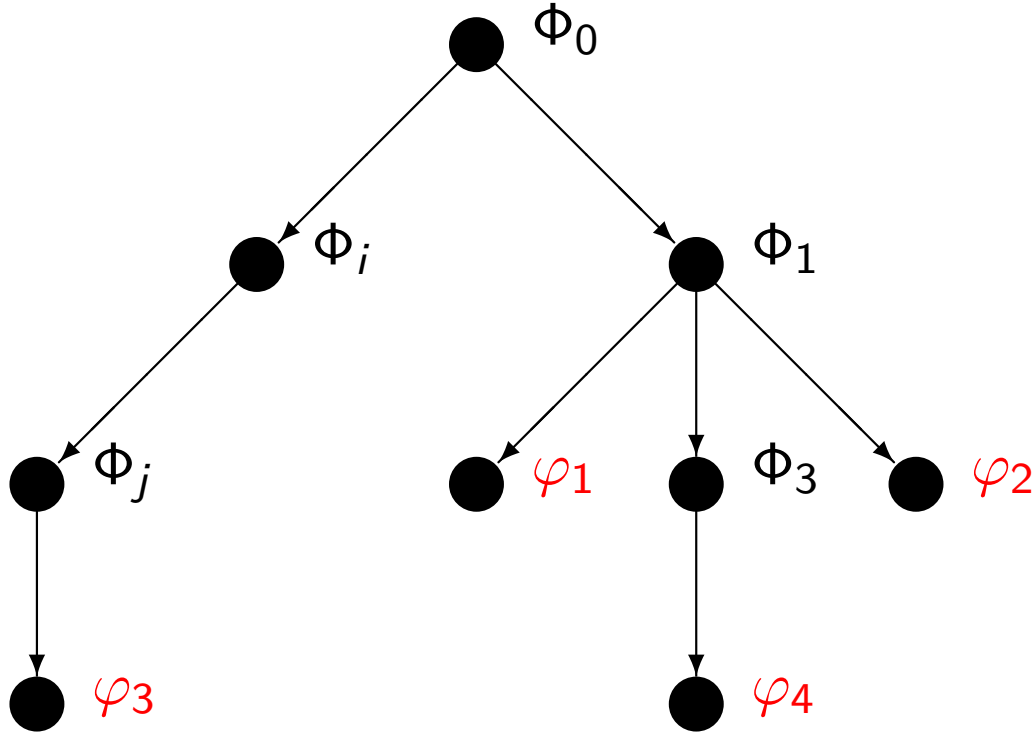
$$\frac{\Phi_1}{\varphi_1,\ \Phi_3,\ \varphi_2}$$ — правило табличного вывода;



ЛОГИКА ХОАРА

Определение вывода в логике Хоара

3) листьями дерева являются формулы логики предикатов.



ЛОГИКА ХОАРА

Определение вывода в логике Хоара

Вывод триплета $\Phi_0 = \varphi_0 \ \{\pi_0\} \ \psi_0$ в логике Хоара называется успешным в интерпретации I, если дерево вывода является конечным, и все его листовые вершины — это истинные в интерпретации I формулы логики предикатов.

ЛОГИКА ХОАРА

Пример

Покажем, что программа

π_0: **while** $\neg(x = y)$
do if $x > y$ **then** $x \Leftarrow x - y$ **else** $y \Leftarrow y - x$ **fi od**

правильно вычисляет наибольший общий делитель двух положительных целых чисел.

Для этого необходимо сформулировать предусловие φ_0 и постусловие ψ_0, соответствующее этому требованию, и построить успешный вывод триплета $\varphi_0 \ \{\pi_0\} \ \psi_0$ в логике Хоара.

ЛОГИКА ХОАРА

Пример

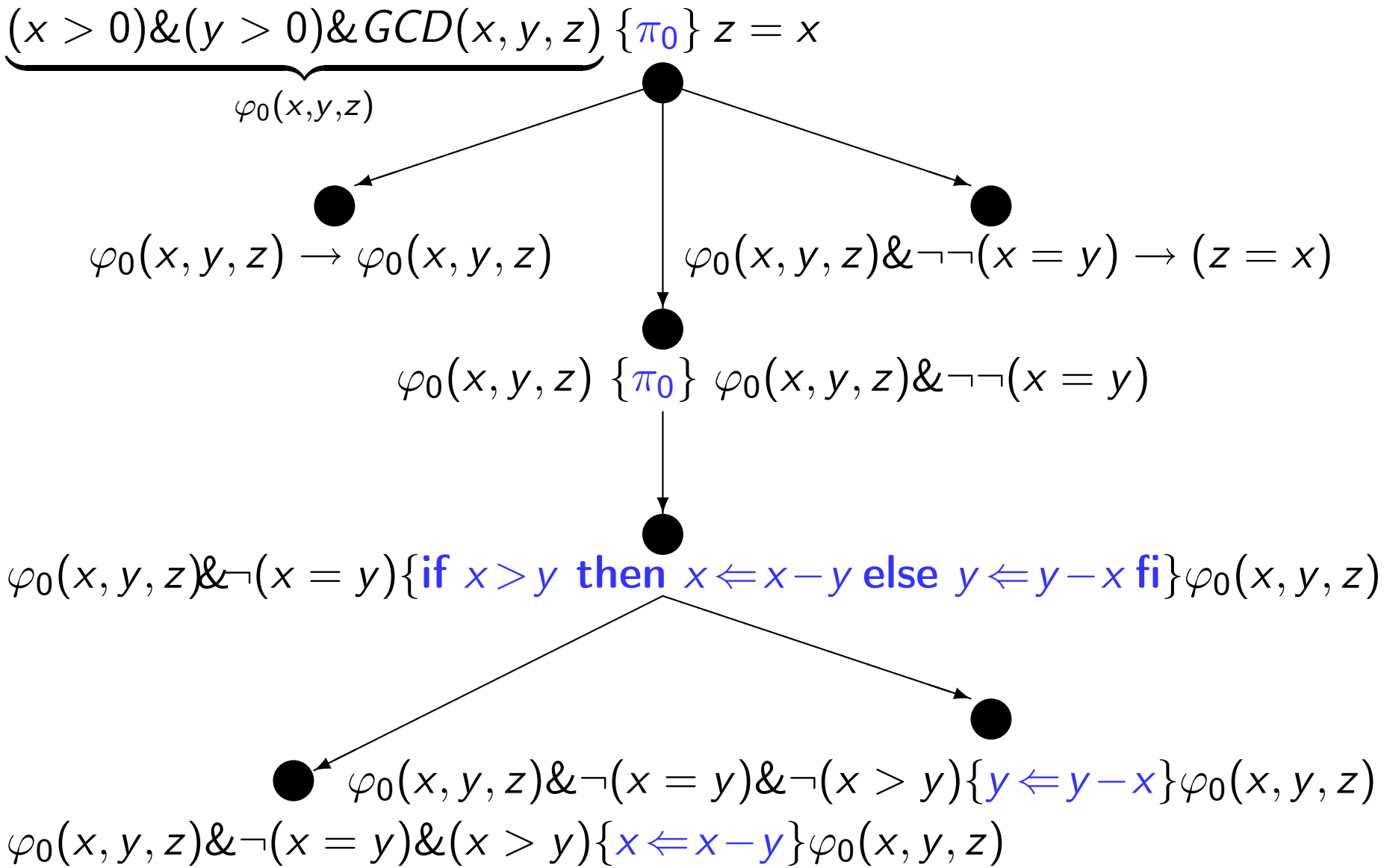
Для удобства обозначений введем некоторые вспомогательные формулы:

$$\begin{array}{lcl} DIV(x,z) & : & \exists u\ (u \times z = x), \\ GCD(x,y,z) & : & DIV(x,z)\ \&\ DIV(y,z)\ \& \\ & & \forall u\ (DIV(x,u) \& DIV(y,u) \rightarrow (u \leq z)). \end{array}$$

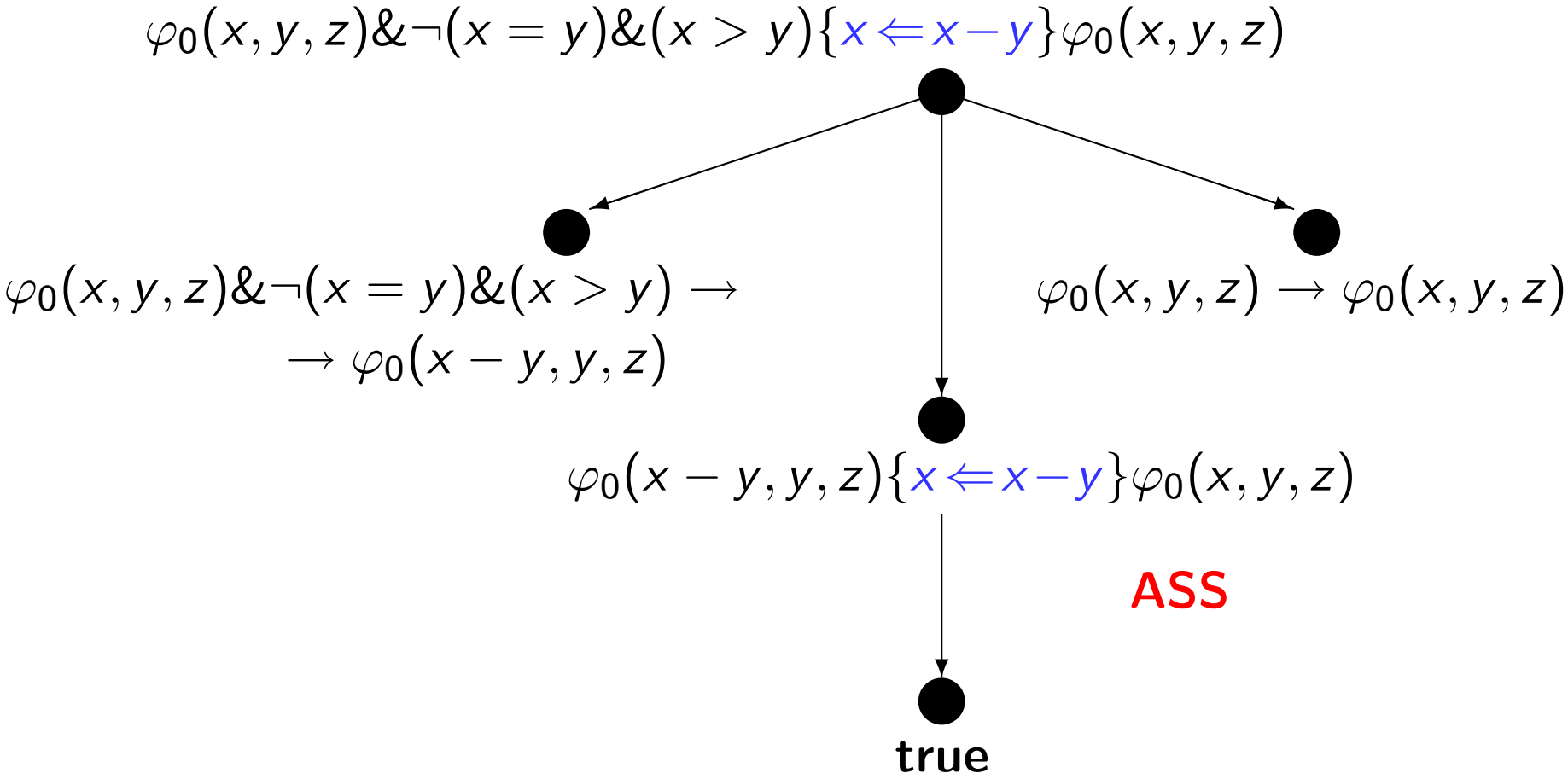
Тогда

$$\begin{array}{lcl} \varphi_0(x,y,z) & : & (x > 0)\ \&\ (y > 0)\ \&\ GCD(x,y,z), \\ \psi_0(x,z) & : & z = x. \end{array}$$

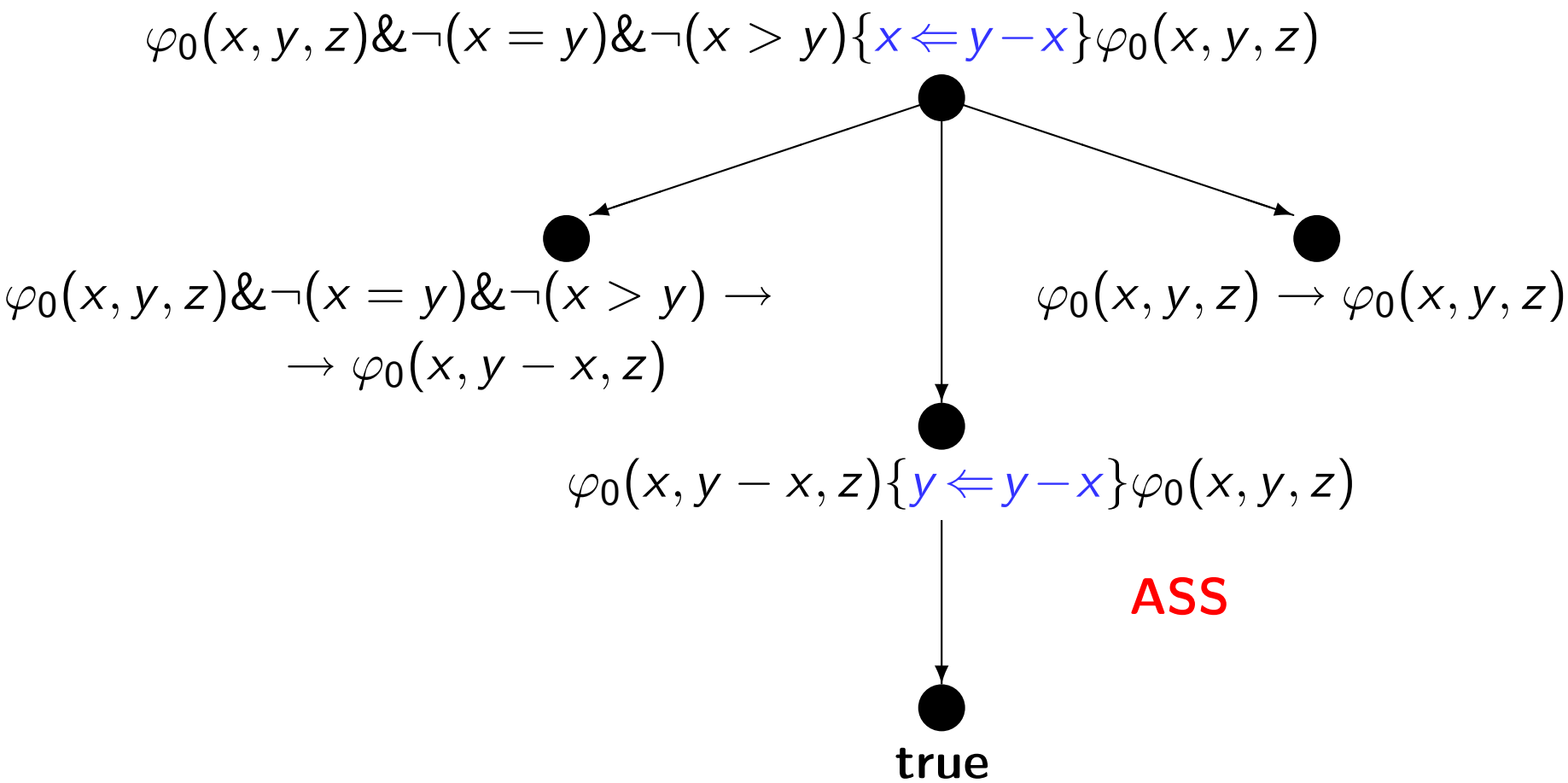
π_0: **while** $\neg(x = y)$
do if $x > y$ **then** $x \Leftarrow x - y$ **else** $y \Leftarrow y - x$ **fi od**



Левая ветвь


$\varphi_0(x,y,z)\&\neg(x=y)\&(x>y)\{x \Leftarrow x-y\}\varphi_0(x,y,z)$
$\varphi_0(x,y,z)\&\neg(x=y)\&(x>y) \rightarrow$
$\rightarrow \varphi_0(x-y,y,z)$
$\varphi_0(x,y,z) \rightarrow \varphi_0(x,y,z)$
$\varphi_0(x-y,y,z)\{x \Leftarrow x-y\}\varphi_0(x,y,z)$
ASS
true

Правая ветвь



ЛОГИКА ХОАРА

Пример

Покажем, что построенный вывод в логике Хоара является успешным для стандартной арифметической интерпретации $I_0 = \langle D_{I_0} = \{0, 1, 2, \dots\}, \{+, -, \times\}, <, >, =, \geq, \leq \rangle$.

Для этого достаточно установить истинность в интерпретации I_0 всех формул, стоящих в листьях построенного вывода.

1. $I_0 \models \varphi(x, y, z) \rightarrow \varphi(x, y, z)$ Очевидно.
2. $I_0 \models \varphi_0(x, y, z) \& \neg\neg(x = y) \rightarrow (z = x)$, т. е.
   $I_0 \models (x > 0) \& (y > 0) \& (x = y) \& GCD(x, y, z) \rightarrow (z = x)$.
   Верно.

ЛОГИКА ХОАРА

Пример

3. $I_0 \models \varphi_0(x, y, z) \& \neg(x = y) \& (x > y) \rightarrow$
   $\rightarrow \varphi_0(x - y, y, z)$, т. е.
   $I_0 \models (x > 0) \& (y > 0) \& (x > y) \& GCD(x, y, z) \rightarrow$
   $\rightarrow (x - y > 0) \& (y > 0) \& GCD(x - y, y, z)$.
   Верно.
4. $I_0 \models \varphi_0(x, y, z) \& \neg(x = y) \& \neg(x > y) \rightarrow$
   $\rightarrow \varphi_0(x, y - x, z)$, т. е.
   $I_0 \models (x > 0) \& (y > 0) \& (y > x) \& GCD(x, y, z) \rightarrow$
   $\rightarrow (x > 0) \& (y - x > 0) \& GCD(x, y - x, z)$.
   Верно.
5. $I_0 \models$ **true**. Очевидно.

Таким образом, все листовые формулы вывода истинны в интерпретации I_0. Значит, вывод триплета $\varphi_0 \ \{\pi_0\} \ \psi_0$ является успешным выводом в интерпретации I_0.

ЛОГИКА ХОАРА

Теорема корректности

Для любой интерпретации I и для любого правила вывода логики Хоара

$$\frac{\Phi}{\Psi}, \quad \frac{\Phi}{\varphi}, \quad \frac{\Phi}{\Psi_1, \Psi_2}, \quad \frac{\Phi}{\varphi, \Psi, \psi},$$

если $I \models \Psi$, $\quad I \models \varphi$, $\quad \begin{cases} I \models \Psi_1, \\ I \models \Psi_2, \end{cases} \quad \begin{cases} I \models \varphi, \\ I \models \Psi, \\ I \models \psi, \end{cases}$

то $I \models \Phi$.

Доказательство.

Рассмотрим поочередно все правила вывода логики Хоара.

ЛОГИКА ХОАРА

Доказательство.

Правило

ASS: $\dfrac{\varphi\{x/t\}\ \{x \Leftarrow t\}\ \varphi}{\textbf{true}}$.

Покажем, что в любой интерпретации I верно

$$I \models \varphi\{x/t\}\ \{x \Leftarrow t\}\ \varphi. \qquad (*)$$

Пусть θ — произвольная оценка переменных, и пусть $I \models \varphi\{x/t\}\theta$.
Тогда согласно операционной семантике императивных программ имеется единственное вычисление

$$\langle x \Leftarrow t,\ \theta\rangle \ \longrightarrow_I\ \langle \varnothing,\ \eta\rangle,$$

и при этом $\eta = \{x/t\}\theta$.
Очевидно, $I \models \varphi\eta$, и это доказывает $(*)$.

ЛОГИКА ХОАРА

Доказательство.

Для остальных правил доказательство корректности проводится по той же схеме, но более изощренно.
Попробуйте завершить доказательство самостоятельно.

□

Следствие.

Если триплет $\varphi\{\pi\}\psi$ имеет успешный в интерпретации I вывод, то программа π частично корректна в интерпретации I относительно предусловия φ и постусловия ψ.

В частности, это означает, что исследованная нами программа вычисления наибольшего общего частично корректна в арифметической интерпретации I_0.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Как автоматизировать верификацию программ?

Для этого нужно выяснить

1. Полна ли система правил вывода логики Хоара?
2. Существует ли алгоритм построения успешного вывода?

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Вопрос о полноте правил вывода Хоара.

На самом деле, здесь не один а три вопроса.

1. Верно ли, что для каждой интерпретации *I* существует система правил вывода, позволяющая для каждого триплета $\Phi = \varphi\{\pi\}\psi$ построить успешный вывод Φ в интерпретации *I* и **доказать его успешность** в случае $I \models \Phi$?

Ответ отрицательный . Следует из теоремы Геделя о неполноте.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Вопрос о полноте правил вывода Хоара.

2. Верно ли, что для каждой интерпретации *I* существует система правил вывода, позволяющая для каждого триплета $\Phi = \varphi\{\pi\}\psi$ построить успешный вывод Φ в интерпретации *I* (**но не гарантирующая доказательства его успешности**) в случае $I \models \Phi$?

Ответ отрицательный . Базовые предикаты сигнатуры σ могут быть недостаточно выразительными для представления всех тех отношений между переменными программы, которые нужны для построения успешного вывода.
В результате не найдется нужных формул φ', ψ' для применения правила

CONS: $$\frac{\varphi\{\pi\}\psi}{\varphi \to \varphi',\ \varphi'\{\pi\}\psi',\ \psi' \to \psi}\ .$$

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Вопрос о полноте правил вывода Хоара.

3. Верно ли, что для некоторых интерпретаций I существует система правил вывода Хоара, которая позволяет для каждого триплета $\Phi = \varphi\{\pi\}\psi$ построить успешный вывод Φ в интерпретации I в случае $I \models \Phi$?

Ответ положительный . Достаточно, чтобы для любого цикла $\pi =$ **while** C **do** π' **od** существовал такой терм t_π, что для любой оценки переменных θ значение терма $t_\pi\theta$ равно $n+1$ тогда и только тогда, когда цикл π в вычислении $\langle \pi,\ \theta \rangle$ совершает n итераций.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Что нужно для построения успешного вывода?

- Необходимо иметь эффективный прувер для проверки истинности формул в разных интерпретациях:

$$I \models \varphi \ .$$

.

CONS: $$\frac{\varphi\{\pi\}\psi}{\varphi \to \varphi',\ \varphi'\{\pi\}\psi',\ \psi' \to \psi}\ ,$$

поскольку неясно, какие формулы φ', ψ' нужно выбирать в каждом случае.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Стратегия вывода в логике Хоара.

Определение

Пусть заданы интерпретация I, императивная программа π и постусловие ψ. Тогда формула φ_0 называется слабейшим предусловием (weakest postcondition) для программы π и постусловия ψ, если

1. $I \models \varphi_0\{\pi\}\psi$,
2. для любой формулы φ, если $I \models \varphi\{\pi\}\psi$, то $I \models \varphi \to \varphi_0$.

Слабейшее предусловие для программы π и постусловия ψ условимся обозначать $wpr(\pi, \psi)$.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Какая польза от слабейшего предусловия?

Теорема

$$I \models \varphi\{\pi\}\psi \iff \begin{cases} I \models \mathit{wpr}(\pi, \psi)\{\pi\}\psi, \\ I \models \varphi \to \mathit{wpr}(\pi, \psi). \end{cases}$$

Таким образом, задача построения успешного вывода сводится к задаче вычисления $\mathit{wpr}(\pi, \psi)$.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

А как вычислять слабейшее предусловие?

Теорема

$wpr(x \Leftarrow t,\ \psi) = \psi\{x/t\},$

$wpr(\pi_1;\ \pi_2,\ \psi) = wpr(\pi_1,\ wpr(\pi_2,\ \psi)),$

$wpr(\textbf{if}\ C\ \textbf{then}\ \pi_1\ \textbf{else}\ \pi_2\ \textbf{fi}, \psi) =$

$$C \& wpr(\pi_1,\ \psi) \vee \neg C \& wpr(\pi_2,\ \psi),$$

Доказательство

Самостоятельно.

Таким образом, для многих операторов (программ) слабейшее предусловие вычисляется автоматически.

АВТОМАТИЧЕСКАЯ ПРОВЕРКА ПРАВИЛЬНОСТИ ПРОГРАММ

Неужели все так просто?

Увы, нет. Главную трудность представляет оператор цикла **while** C **do** π **od**. Единственный способ верифицировать этот оператор — это воспользоваться производным правилом:

$$\textbf{WHILE-GEN:}\quad \frac{\varphi\ \{\textbf{while } C \textbf{ do } \pi \textbf{ od}\}\ (\psi)}{\varphi \to \chi,\ (\chi \& C)\ \{\pi\}\ \chi,\ (\chi \& \neg C) \to \psi}\ .$$

Это правило требует введения вспомогательной формулы χ, которая называется инвариантом цикла . Инвариант цикла зависит от программы π и условия C.
Автоматическая генерация инвариантов цикла — это ключевая задача в решении проблемы автоматической верификации программ.

КОНЕЦ ЛЕКЦИИ 20.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

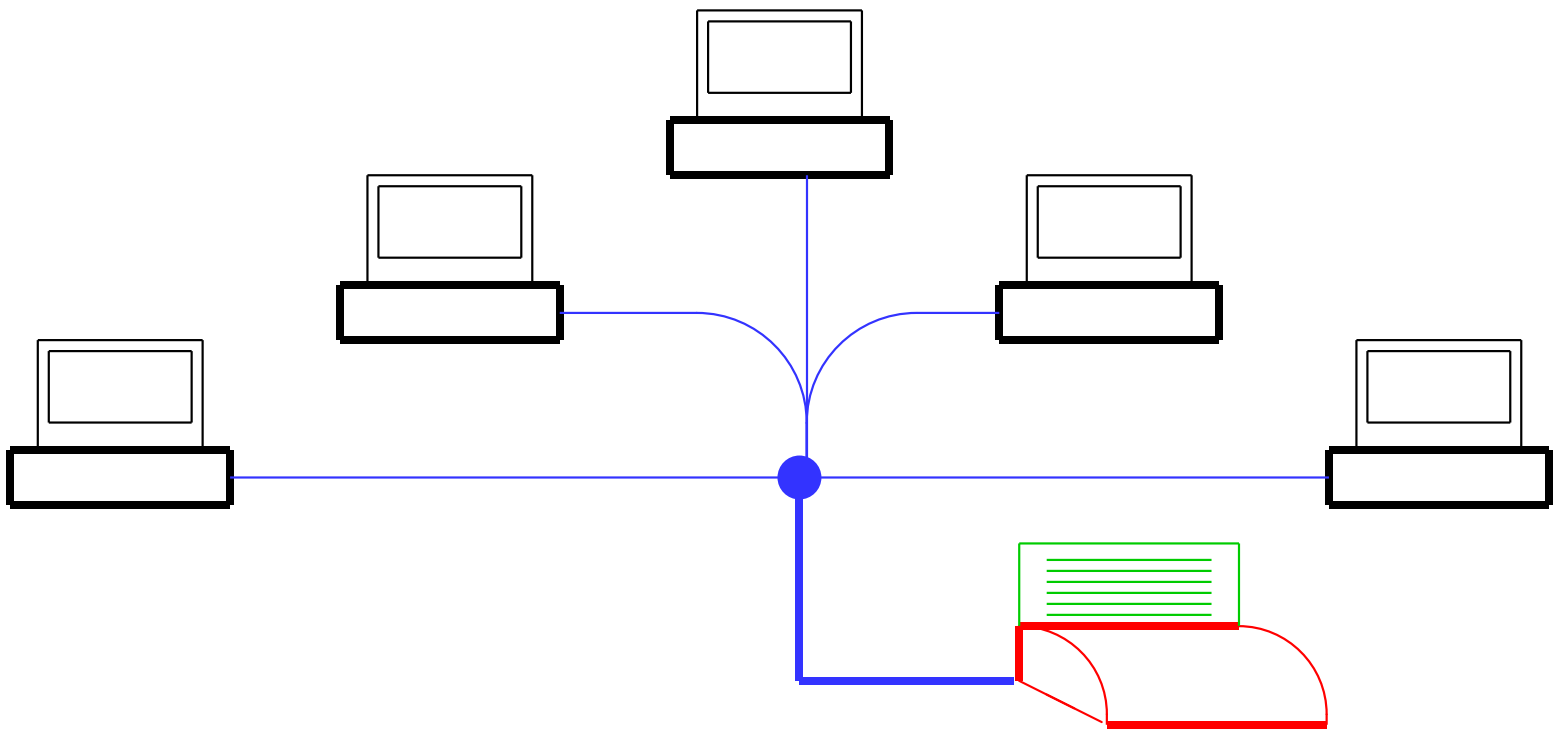
Лекция 21.

Верификация распределенных программ. Логика линейного времени PLTL. Размеченные системы переходов. Задача верификации моделей программ.

ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Задача

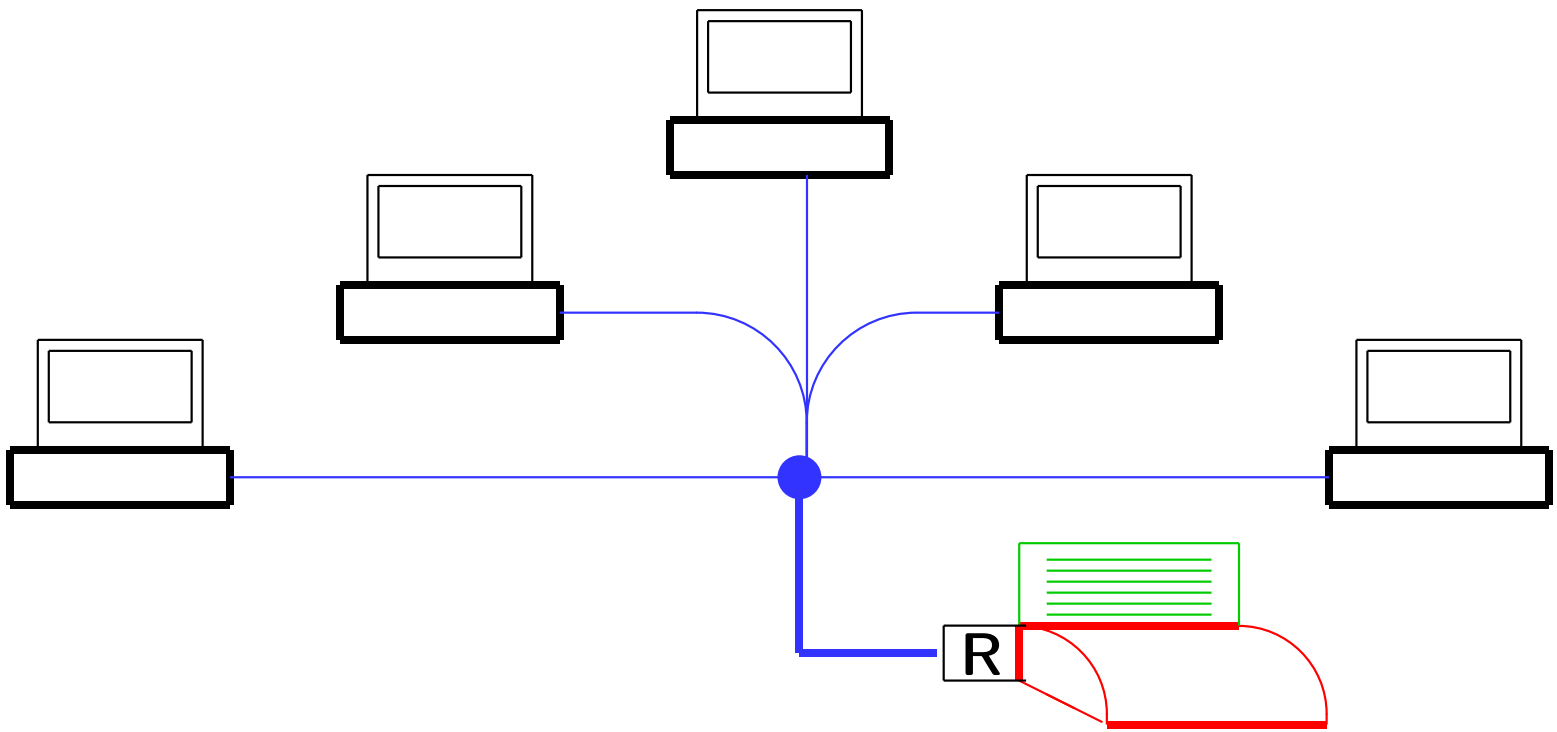
Имеется несколько компьютеров и только один принтер. Ни один компьютер не осведомлен о существовании других компьютеров. Как правильно организовать их взаимодействие, чтобы все они могли пользоваться этим принтером?



ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Задача

Предполагается также, что у принтера есть единственный однобитовый регистр **R**, общедоступный для считывания и записи. Этот регистр может находится в одном из двух состояний — *busy* (принтер занят) и *free* (принтер свободен).



ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Прежде чем писать программу (драйвер), обеспечивающую взаимодействие каждого компьютера с принтером, нужно сформулировать требования, предъявляемые к этой программе.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;
2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати;
3. Компьютер, завершивший печать, должен когда-нибудь освободить принтер;
4. Данные на печать всегда передает не более чем один компьютер.

А какие еще требования разумно предъявить к нашему драйверу?

ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Для связи с принтером программист предложил снабдить каждый компьютер одной и той же программой

L_1: **while** $R \neq$ *free* **do wait od** ;

L_2: R = *busy*;

L_3: **output**(X,*printer*);

L_4: R = *free*;

Это простая и разумная программа.
Но будет ли система компьютеров, снабженных этой программой, вести себя в соответствии с указанными требованиями?

ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Если мы имеем дело с **последовательной программой**, то наиболее простой способ проверки ее правильности — это тестирование. Тестирование, вообще говоря, не гарантирует того, что все вычисления являются правильными, но оно позволяет, по крайней мере, убедиться в том, что программа ведет себя правильно на тестовых примерах.

Но если мы занимаемся верификацией **распределенных программ**, состоящих из нескольких процессов, работающих на независимых вычислительных устройствах, то тестирование не позволяет проверить правильность поведения распределенной системы даже на тестовых примерах.

Почему?

ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Предположим, что на заданных входных данных (тестовом примере) каждый из 20 процессов распределенной системы выполняет всего лишь одно действие. Тогда вычисление системы может быть физически реализовано $20! > 2^{18}$ способами в зависимости от той последовательности, в которой будут завершаться выполнения этих действий. Ясно, что рассмотреть все эти выполнения практически невозможно.

А вместе с тем одни те же действия, выполненные в разной последовательности, могут приводить к разным результатам:

«Наполнить бассейн водой» $\|$ «Прыгнуть в бассейн с вышки»

Как же проверять правильность распределенных систем?

ВЕРИФИКАЦИЯ РАСПРЕДЕЛЕННЫХ ПРОГРАММ

Верификацию распределенных систем нужно автоматизировать. Это можно сделать, например, так.

1. Выбрать логический язык $\mathcal{L}$, на котором можно описывать требования, предъявляемые к программе. Представить эти требования в виде формул $\varphi_1, \ldots, \varphi_n$.
2. Выбрать математическую модель M, адекватно представляющую все вычисления программы. Модель должна быть устроен так, чтобы каждое вычисление I в модели M являлось интерпретацией языка $\mathcal{L}$.
3. Проверить выполнимость формул $\varphi_1, \ldots, \varphi_n$ на всех вычислениях модели M. Для проверки выполнимости формул языка $\mathcal{L}$ на модели программы M должен быть разработан эффективный алгоритм.

Такой подход к проверке правильности программ назвается верификацией моделей программ (англ. model-checking).

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

При верификации распределенных систем, как правило, требуется проверить, что в каждом вычислении системы некоторые события (выполнение того или иного действия, прием/передача сообщений и пр.) происходят в определенной последовательности.

Каждое событие *event* можно охарактеризовать булевой переменной (0-местным предикатом) p_{event}, которая принимает значение **true** в том и только том случае, когда осуществляется событие *event*. Таким образом, в логическом языке $\mathcal{L}$ не нужны предметные переменные, термы, кванторы.

Однако осуществимость событий (значения булевых пременных p_{event}) изменяется со временем. Значит, в логическом языке $\mathcal{L}$ должен быть явно учтен феномен времени.

Таким образом, для описания требований, которые предъявляются к распределенной системе, достаточно воспользоваться языком пропозициональной темпоральной логики линейного времени (PLTL).

Исторические сведения

1941

АМИР ПНУЭЛИ

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Синтаксис PLTL

В PLTL наряду с булевыми логическими связками для описания причинно-следственной зависимости событий во времени применяются темпоральные операторы

- **X** (ne**X**ttime) «в следующий момент времени»;
- **F** (sometime in **F**uture) «когда-то в будущем»;
- **G** (**G**lobally) «всегда в будущем»;
- **U** (**U**ntil) «до тех пор пока»;
- **R** (**R**elease) «высвободить, открепить».

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Пусть задано множество булевых переменных $\mathcal{AP} = \{p_1, p, \ldots, p_n, \ldots\}$ (будем называть их атомарными высказываниями).

Синтаксис PLTL

Формула PLTL — это

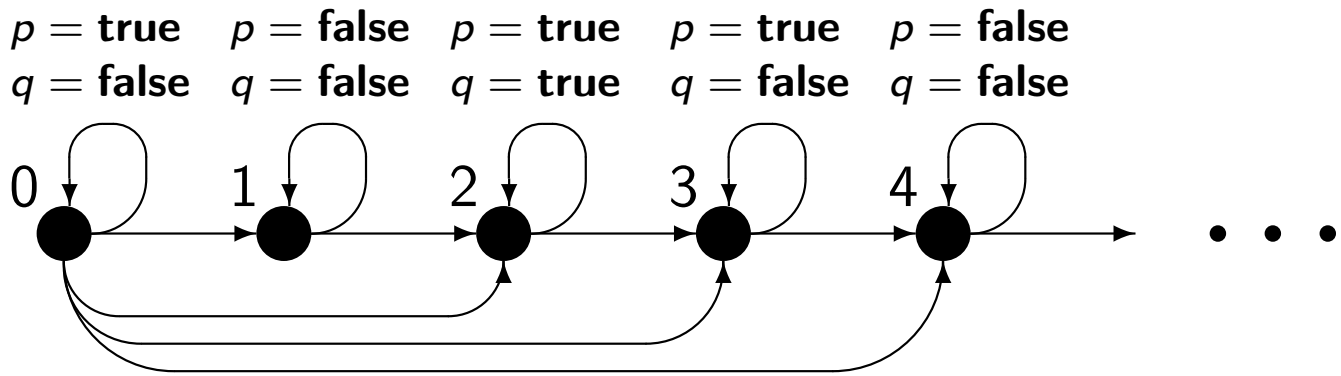
p_i,	если $p_i \in \mathcal{AP}$;
$(\varphi \& \psi)$,	если φ и ψ — формулы;
$(\varphi \vee \psi)$,	
$(\varphi \to \psi)$,	
$(\neg\varphi)$,	
$(\mathbf{X}\varphi)$,	«в следующий момент будет верно φ»;
$(\mathbf{F}\varphi)$,	«когда-то в будущем будет верно φ»;
$(\mathbf{G}\varphi)$,	«всегда верно φ»;
$(\varphi \mathbf{U} \psi)$,	«φ остается верной, пока не станет верной ψ»;
$(\varphi \mathbf{R} \psi)$,	«ψ может перестать быть верной только после того, как станет верной φ».

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Семантика PLTL

Интерпретация PLTL — это темпоральная модель Крипке $I = \langle \mathbb{N}, \leq, \xi \rangle$, где

- $\mathbb{N} = \{0, 1, 2, \dots\}$ — множество моментов времени;
- $\leq$ — отношение нестрогого линейного порядка на $\mathbb{N}$;
- $\xi : \mathbb{N} \times \mathcal{AP} \to \{\mathbf{true}, \mathbf{false}\}$ — оценка атомарных высказываний на шкале времени.



ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Семантика PLTL

Интерпретация $I = \langle \mathbb{N}, \leq, \xi \rangle$ — это вычислительная трасса программы, и в этой трассе

- $\mathbb{N} = \{0, 1, 2, \dots\}$ — это последовательность состояний вычисления, линейно упорядоченная отношением переходов $\leq$;
- оценка $\xi : \mathbb{N} \times \mathcal{AP} \to \{\textbf{true}, \textbf{false}\}$ указывает, какие события происходят в те или иные моменты времени.

Формулы PLTL — это утверждения о том, в какой последовательности должны происходить события по ходу вычислений программ.

Чтобы оценивать, в какой мере вычислительная трасса (интерпретация) удовлетворяет заданному требованию (формуле PLTL), определим отношение выполнимости формул PLTL в темпоральных интерпретациях.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

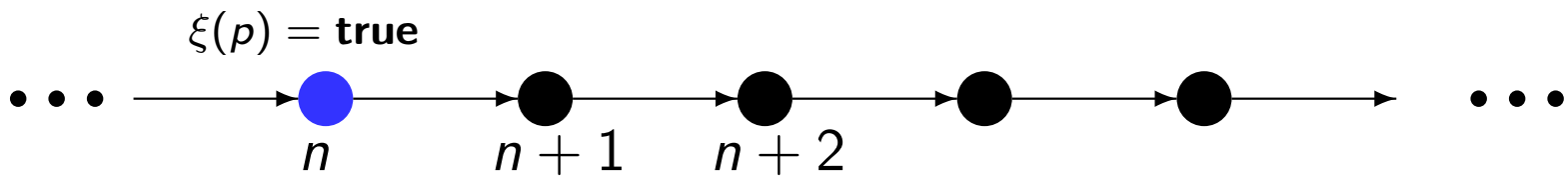
Семантика PLTL

Пусть $I = \langle \mathbb{N}, \leq, \xi \rangle$ — темпоральная интерпретация (вычислительная трасса), $n \in \mathbb{N}$ — момент времени (состояние вычисления), φ — формула PLTL.

Тогда отношение выполнимости $I, n \models \varphi$ формулы φ в момент времени n в интерпретации I определяется так.

1. Если $\varphi = p$, $p \in \mathcal{AP}$ (т. е. φ — атомарное высказывание), то

$$I, n \models \varphi \iff \xi(p) = \textbf{true}.$$

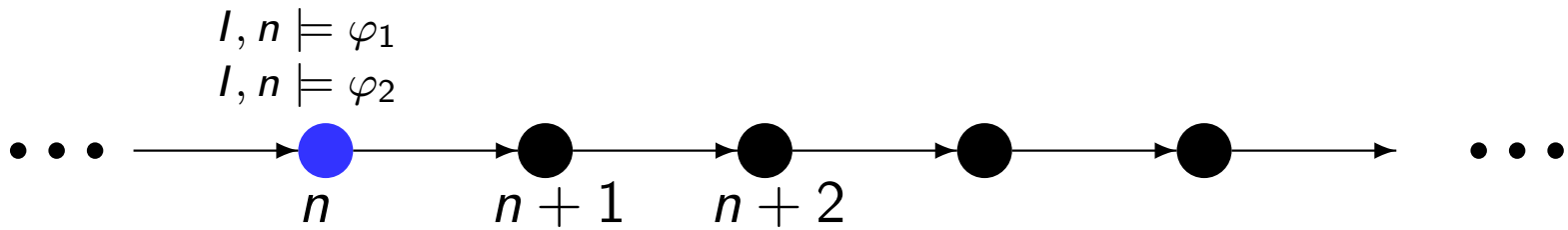


ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Семантика PLTL

2. Если $\varphi = \varphi_1 \& \varphi_2$, то

$$I, n \models \varphi \iff I, n \models \varphi_1 \text{ и } I, n \models \varphi_2.$$



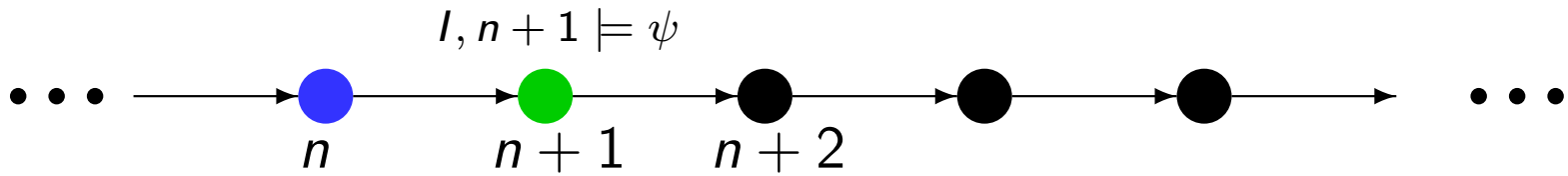
Для формул вида $\varphi_1 \vee \varphi_2$, $\varphi_1 \to \varphi_2$, $\neg\varphi_1$ отношение выполнимости в темпоральной модели определяется точно так же, как в классической логике предикатов.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Семантика PLTL

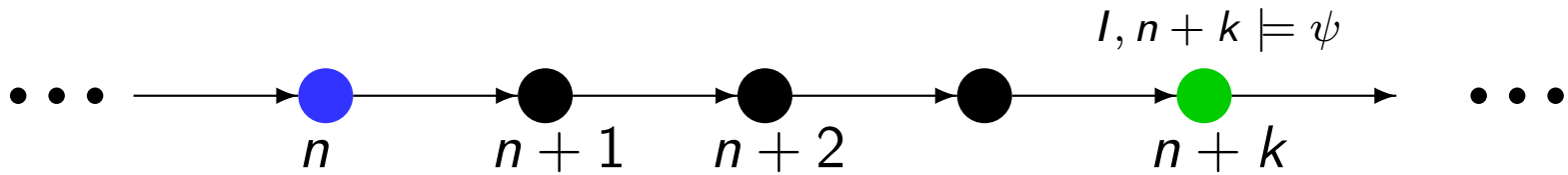
3. Если $\varphi = \mathbf{X}\psi$, то

$$l, n \models \varphi \iff l, n+1 \models \psi.$$



4. Если $\varphi = \mathbf{F}\psi$, то

$l, n \models \varphi \iff$ существует такое $k, k \geq 0,$ что $l, n+k \models \psi.$

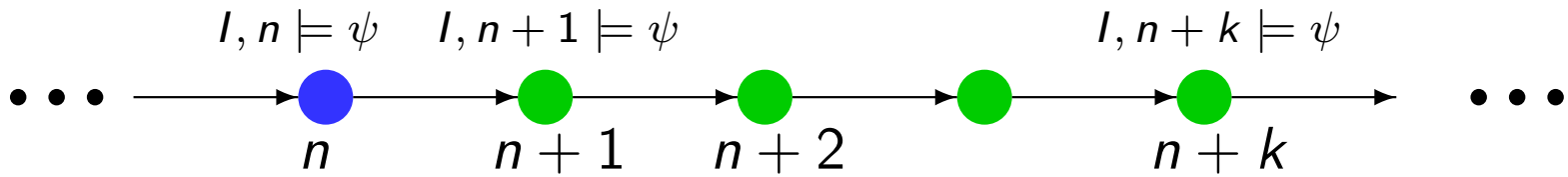


ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Семантика PLTL

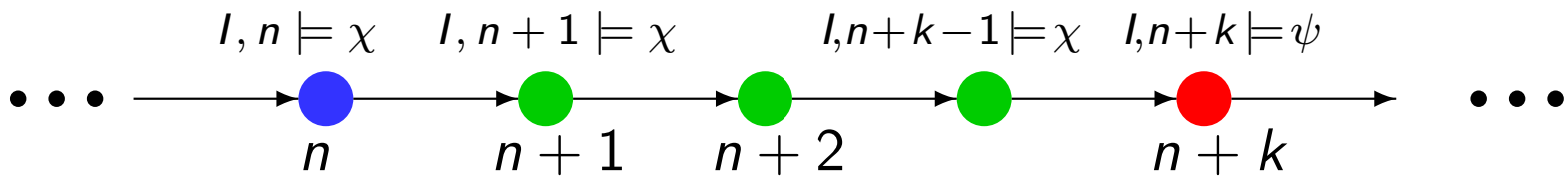
5. Если $\varphi = \mathbf{G}\psi$, то

$$I, n \models \varphi \iff \text{для любого } k, k \geq 0, \text{ верно } I, n+k \models \psi.$$



6. Если $\varphi = \chi\mathbf{U}\psi$, то

$$I, n \models \varphi \iff \text{существует такое } k, k \geq 0, \text{ что } I, n+k \models \psi,$$
$$\text{и для любого } i, 0 \leq i < k, \text{ верно } I, n+i \models \chi.$$



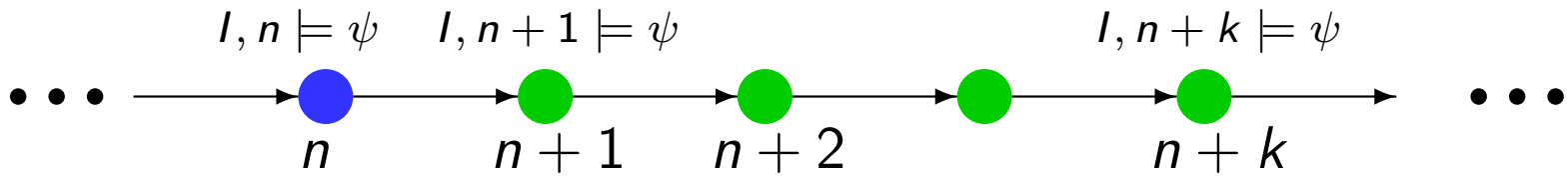
ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Семантика PLTL

7. Если $\varphi = \chi \mathbf{R} \psi$, то

$$I, n \models \varphi \iff \text{либо для любого } k, k \geq 0, \text{ верно } I, n+k \models \psi,$$

,



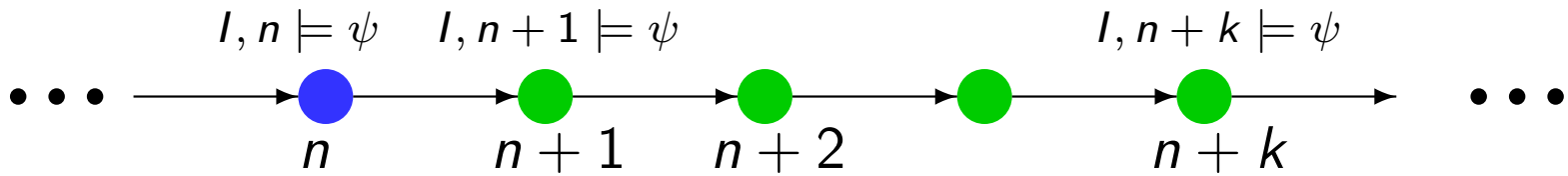
ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

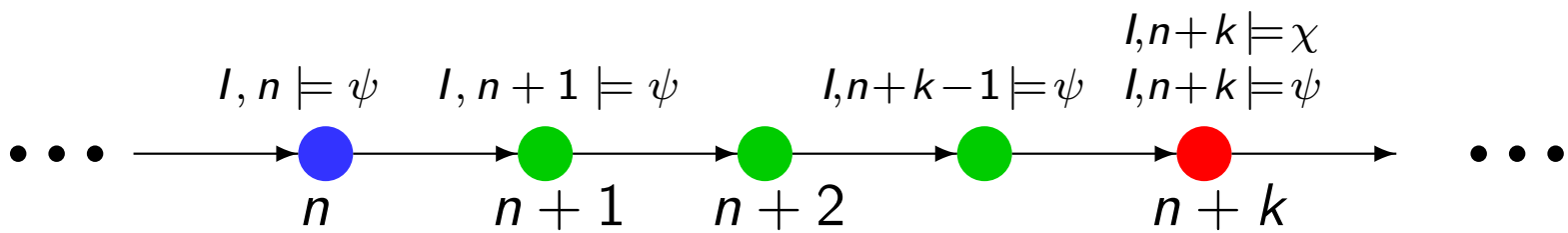
Семантика PLTL

7. Если $\varphi = \chi \mathbf{R} \psi$, то

$$I, n \models \varphi \iff$$ либо для любого $k, k \geq 0$, верно $I, n + k \models \psi$,

либо существует такое $k, k \geq 0$, что $I, n + k \models \chi$,
и для любого $i, 0 \leq i \leq k$, верно $I, n + i \models \psi$.





ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Будем называть формулу PLTL φ

- выполнимой в интерпретации I, если верно $I, 0 \models \varphi$ (обозначается $I \models \varphi$);
- PLTL-общезначимой , если для любой интерпретации I верно $I \models \varphi$ (обозначается $\models \varphi$).

Чтобы облегчить запись формул и избавиться от лишних скобок, условимся, что
одноместные темпоральные операторы **X**, **F**, **G** обладают таким же приоритетом, как отрицание $\neg$,
а двухместные темпоральные операторы **U**, **R** имеют наивысший приоритет среди двухместных связок.

Таким образом, запись

$$\mathbf{X}p_1\mathbf{U}p_2\&\mathbf{F}p_3 \rightarrow \neg p_1\mathbf{R}p_2$$

обозначает формулу

$$\big(((\mathbf{X}p_1)\mathbf{U}p_2)\&(\mathbf{F}p_3)\big) \rightarrow \big((\neg p_1)\mathbf{R}p_2\big).$$

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Равносильные формулы

Темпоральные операторы PLTL связаны друг с другом определенными соотношениями (равносильностями). Вот наиболее важные из соотношений равносильности.

Законы двойственности.

1. $\models \neg \mathbf{X}\varphi \equiv \mathbf{X}\neg\varphi$;
2. $\models \neg \mathbf{F}\varphi \equiv \mathbf{G}\neg\varphi$;
3. $\models \neg \mathbf{G}\varphi \equiv \mathbf{F}\neg\varphi$;
4. $\models \neg(\varphi \mathbf{U} \psi) \equiv \neg\varphi \mathbf{R} \neg\psi$;
5. $\models \neg(\varphi \mathbf{R} \psi) \equiv \neg\varphi \mathbf{U} \neg\psi$.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Равносильные формулы

Темпоральные операторы PLTL связаны друг с другом определенными соотношениями (равносильностями).
Вот наиболее важные из соотношений равносильности.

Законы двойственности.

1. $\models \neg \mathbf{X}\varphi \equiv \mathbf{X}\neg\varphi$;
2. $\models \neg \mathbf{F}\varphi \equiv \mathbf{G}\neg\varphi$;
3. $\models \neg \mathbf{G}\varphi \equiv \mathbf{F}\neg\varphi$;
4. $\models \neg(\varphi \mathbf{U} \psi) \equiv \neg\varphi \mathbf{R} \neg\psi$;
5. $\models \neg(\varphi \mathbf{R} \psi) \equiv \neg\varphi \mathbf{U} \neg\psi$.

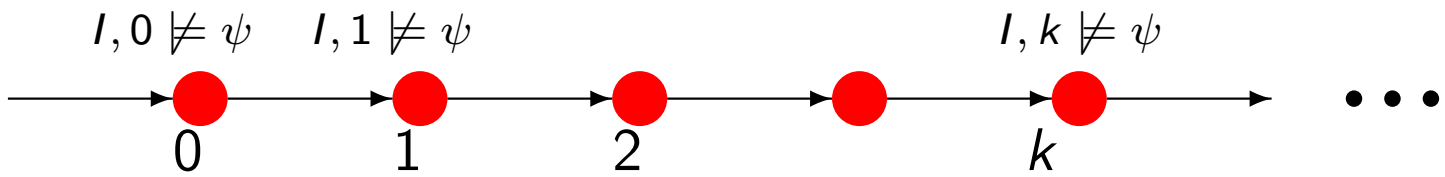
Доказательство. Следует непосредственно из определения отношения выполнимости.
Покажем справедливость соотношения 4).

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

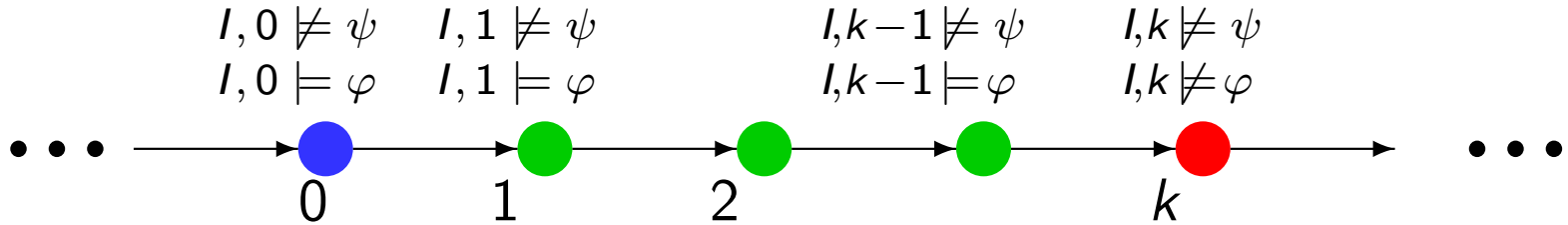
Доказательство.

Пусть $I, 0 \models \neg(\varphi \mathbf{U} \psi)$, т. е. $I, 0 \not\models \varphi \mathbf{U} \psi$. Тогда согласно определению отношения выполнимости для оператора **U** верно хотя бы одно из двух:

1. либо для любого k, $k \geq 0$, верно $I, k \not\models \psi$



2. либо существует такое k, $k \geq 0$, что $I, k \not\models \psi$, $I, k \not\models \varphi$ и при этом для любого i если $0 \leq i < k$, то $I, k \not\models \psi$, $I, k \models \varphi$

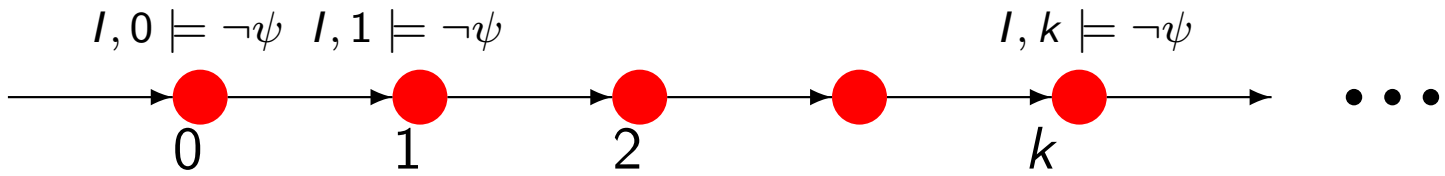


ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

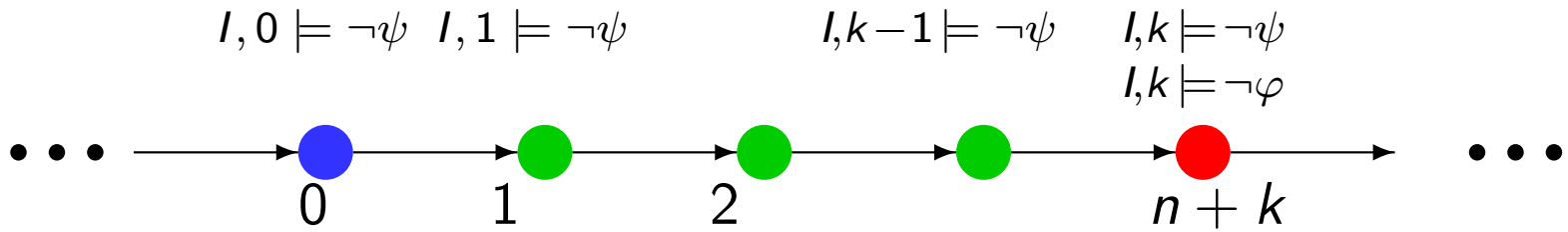
Доказательство.

Но это означает, что верно хотя бы одно из двух:

1. либо для любого k, $k \geq 0$, верно $I, k \models \neg\psi$



2. либо существует такое k, $k \geq 0$, что $I, k \models \neg\psi$, $I, k \models \neg\varphi$ и при этом для любого i если $0 \leq i < k$, то $I, k \models \neg\psi$



А это как раз и означает, что $I, 0 \models (\neg\varphi)\mathbf{R}(\neg\psi)$.

Значит, $I \models \neg(\varphi\mathbf{U}\psi) \rightarrow \neg\varphi\mathbf{R}\neg\psi$ для любой интерпретации I.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Доказательство.

Проводя аналогичные рассуждения покажите самостоятельно , что верно соотношение

$$I \models \neg\varphi \mathbf{R} \neg\psi \rightarrow \neg(\varphi \mathbf{U} \psi),$$

а также остальные законы двойственности темпоральных операторов.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Равносильные формулы

Законы взаимной зависимости.

1. $\models \mathbf{F}\varphi \equiv \neg\mathbf{G}\neg\varphi$;
2. $\models \mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$;
3. $\models \varphi\mathbf{U}\psi \equiv \neg(\neg\varphi\mathbf{R}\neg\psi)$;
4. $\models \varphi\mathbf{R}\psi \equiv \neg(\neg\varphi\mathbf{U}\neg\psi)$;
5. $\models \mathbf{F}\varphi \equiv \mathbf{true}\ \mathbf{U}\varphi$;
6. $\models \mathbf{G}\varphi \equiv \mathbf{false}\ \mathbf{R}\varphi$.

Доказательство. Самостоятельно.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Равносильные формулы

Законы неподвижной точки.

1. $\models \mathbf{F}\varphi \equiv \varphi \vee \mathbf{XF}\varphi$;
2. $\models \mathbf{G}\varphi \equiv \varphi \,\&\, \mathbf{XG}\varphi$;
3. $\models \varphi\mathbf{U}\psi \equiv \psi \vee (\varphi \,\&\, \mathbf{X}(\varphi\mathbf{U}\psi)$;
4. $\models \varphi\mathbf{R}\psi \equiv \psi \,\&\, (\varphi \vee \mathbf{X}(\varphi\mathbf{R}\psi)$.

Доказательство. Самостоятельно.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Равносильные формулы

А какие законы дистрибутивности верны?

1. $\models \mathbf{F}(\varphi \vee \psi) \equiv$???;
2. $\models \mathbf{G}(\varphi \vee \psi) \equiv$???;
3. $\models \mathbf{F}(\varphi \& \psi) \equiv$???;
4. $\models \mathbf{G}(\varphi \& \psi) \equiv$???;
5. $\models \varphi \mathbf{U}(\psi \vee \chi) \equiv$???;
6. $\models \varphi \mathbf{U}(\psi \& \chi) \equiv$???
7. $\models (\varphi \vee \chi)\mathbf{U}\chi \equiv$???;
8. $\models (\varphi \& \chi)\mathbf{U}\chi \equiv$???.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Выразительные возможности PLTL

А теперь посмотрим, насколько адекватно и просто можно выражать при помощи PLTL те требования, которые предъявляются к поведению распределенных систем программ.

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят;
2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати;
3. Компьютер, завершивший печать, должен когда-нибудь освободить принтер;
4. Данные на печать всегда передает не более чем один компьютер.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Выразительные возможности PLTL

Введем следующие атомарные высказывания, соответствующие основным событиям вычислений программы:

1. try_i — i-ый компьютер собирается отправить данные на печать;
2. pr_i — i-ый компьютер передает данные на печать;
3. *free* — принтер свободен;
4. *busy* — принтер занят.

Пусть имеется система, состоящая из 2 компьютеров.
Тогда все перечисленные требования выражаются формулами PLTL так.

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Выразительные возможности PLTL

1. Всякий раз, когда принтер свободен и хотя бы один компьютер собирается отправить данные на печать, принтер будет рано или поздно занят:

$$\varphi_1 = \mathbf{G}((\mathit{free} \;\&\; (\mathit{try}_1 \;\vee\; \mathit{try}_2)) \;\rightarrow\; \mathbf{F}\mathit{busy}) \;;$$

2. Всякий раз, после того как принтер оказался занят, он должен когда-нибудь приступить к печати:

$$\varphi_2 = \mathbf{G}(\mathit{free} \;\&\; \mathbf{X}\mathit{busy} \;\rightarrow\; \mathbf{XF}(\mathit{pr}_1 \;\vee\; \mathit{pr}_2)) \;;$$

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Выразительные возможности PLTL

3. Компьютер, завершивший печать, должен когда-нибудь освободить принтер:

$$\varphi_{3i} = \mathbf{G}(pr_i \ \& \ \mathbf{X}\neg pr_i \ \rightarrow \ \mathbf{XF}free) \ ;$$

4. Данные на печать всегда передает не более чем один компьютер:

$$\varphi_4 = \mathbf{G}(\neg(pr_1 \ \& \ pr_2)) \ .$$

ЛОГИКА ЛИНЕЙНОГО ВРЕМЕНИ PLTL

Выразительные возможности PLTL

5. До тех пор пока хотя бы один компьютер отправляет данные на печать, принтер остается занятым:

$$\varphi_5 = \mathbf{G}((\neg(pr_1 \ \vee \ pr_2) \ \mathbf{R} \ busy) \ ;$$

или может быть это лучше выразить так:

$$\varphi_5' = \mathbf{G}(busy \ \mathbf{R} \ (\neg(pr_1 \ \vee \ pr_2))) \ ?$$

или может быть так:

$$\varphi_5'' = \mathbf{G}((pr_1 \ \vee \ pr_2) \ \rightarrow \ busy) \ ?$$

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

Теперь мы можем использовать PLTL в качестве формального языка спецификации программ.
Чтобы иметь возможность проверить, удовлетворяют ли все вычисления распределенной системы программ заданным спецификациям, которые представлены формулами PLTL, нужно определить математическую модель программ.

При разработке математической модели программ нужно стремиться к тому, чтобы

- каждое вычисление распределенной системы программ представляло собой темпоральную интерпретацию;
- вычисления программ в предложенной математической модели соответствовали реальному поведению вычислительных устройств, выполняющих эти программы;
- модель программ имела простое устройство.

В качестве такой модели программ мы будем использовать размеченные системы переходов .

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

Определение LTS

Размеченная система переходов (LTS, Labelled Transition System) — это пятерка $\langle \mathcal{AP}, S, S_0, \longrightarrow, \rho \rangle$, в которой

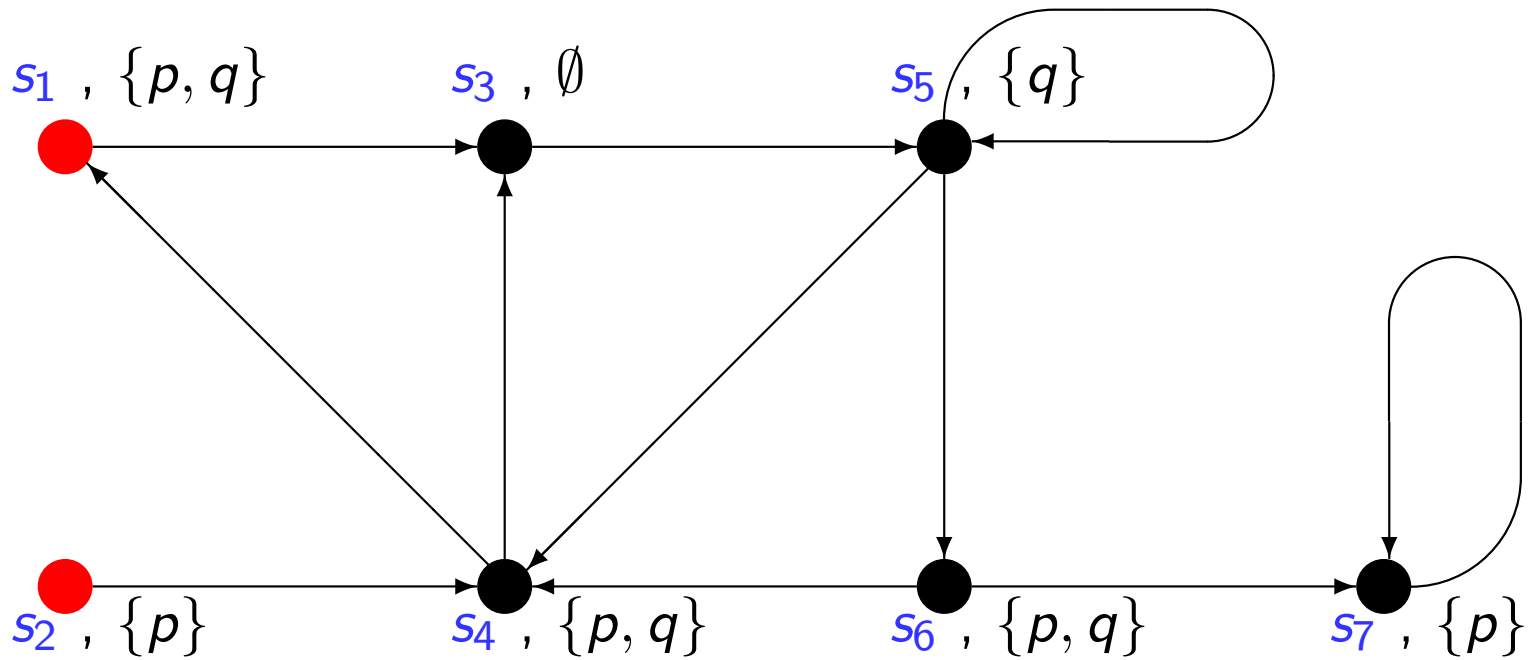
1. $\mathcal{AP}$ — множество атомарных высказываний;
2. S — непустое множество состояний вычислений;
3. S_0, $S_0 \subseteq S$, — непустое подмножество начальных состояний;
4. $\longrightarrow \subseteq S \times S$, — **тотальное** отношение переходов, тотальность отношения $\longrightarrow$ означает, что для любого состояния $s, s \in S$, существует такое состояние s', что $s \longrightarrow s'$ (т. е. из любого состояния можно сделать хотя бы один переход);
5. $\rho : S \rightarrow 2^{\mathcal{AP}}$ — функция разметки, приписывающая каждому состоянию вычислений $s, s \in S$, множество $\rho(s)$, $\rho(s) \subseteq \mathcal{AP}$, всех тех атомарных высказываний, которые являются истинными в состоянии s.

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

Пример LTS

$$M = \langle \mathcal{AP}, S, S_0, T, \rho \rangle$$

$$\mathcal{AP} = \{p, q\}, \quad S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}, \quad S_0 = \{s_1, s_2\}$$



РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS и PLTL-интерпретации

Трассой в LTS $M = \langle \mathcal{AP}, S, S_0, \longrightarrow, \rho \rangle$ называется всякая бесконечная последовательность состояний

$$tr = s_{i_0}, s_{i_1}, \ldots, s_{i_n}, s_{i_{n+1}}, \ldots, \qquad (*)$$

в которой для любого $n, n > 0$, верно $(s_{i_n} \longrightarrow s_{i_{n+1}})$.

Если s_{i_0} — начальное состояние, $s_{i_0} \in S_0$, то трасса tr называется начальной трассой.

Запись $Tr(M)$ обозначает множество всех трасс LTS M, а запись $Tr_0(M)$ — множество всех начальных трасс LTS M.

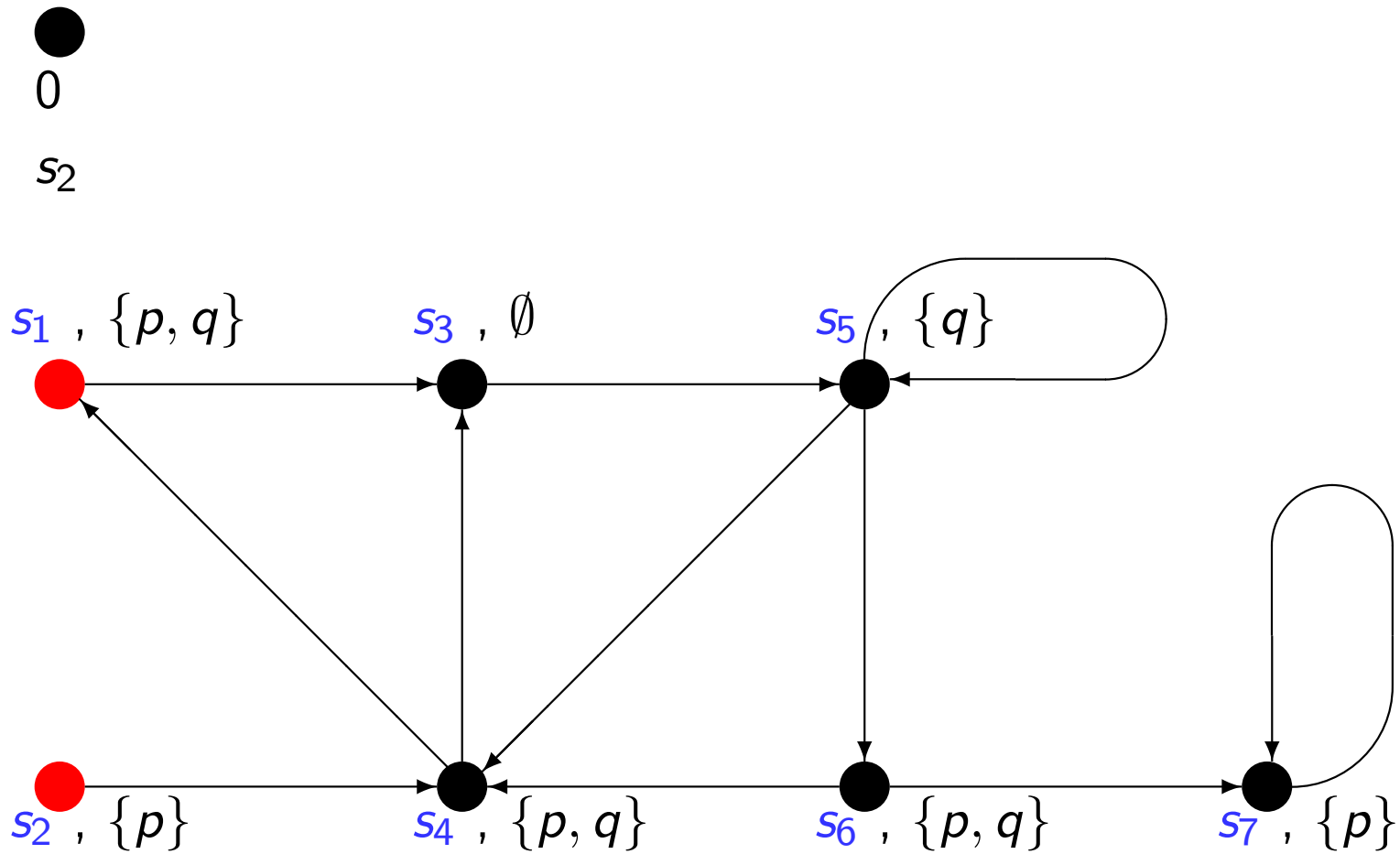
Каждой трассе $tr \in Tr(M)$ вида $(*)$ сопоставим PLTL-интерпретацию $I(tr) = \langle \mathbb{N}, \leq, \xi \rangle$, в которой для любого $n, n \geq 0$, и p, $p \in \mathcal{AP}$, верно соотношение

$$\xi(n, p) = \textbf{true} \iff p \in \rho(s_{i_n}) \ .$$

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

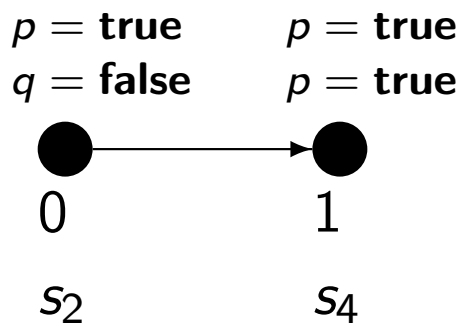
Пример LTS

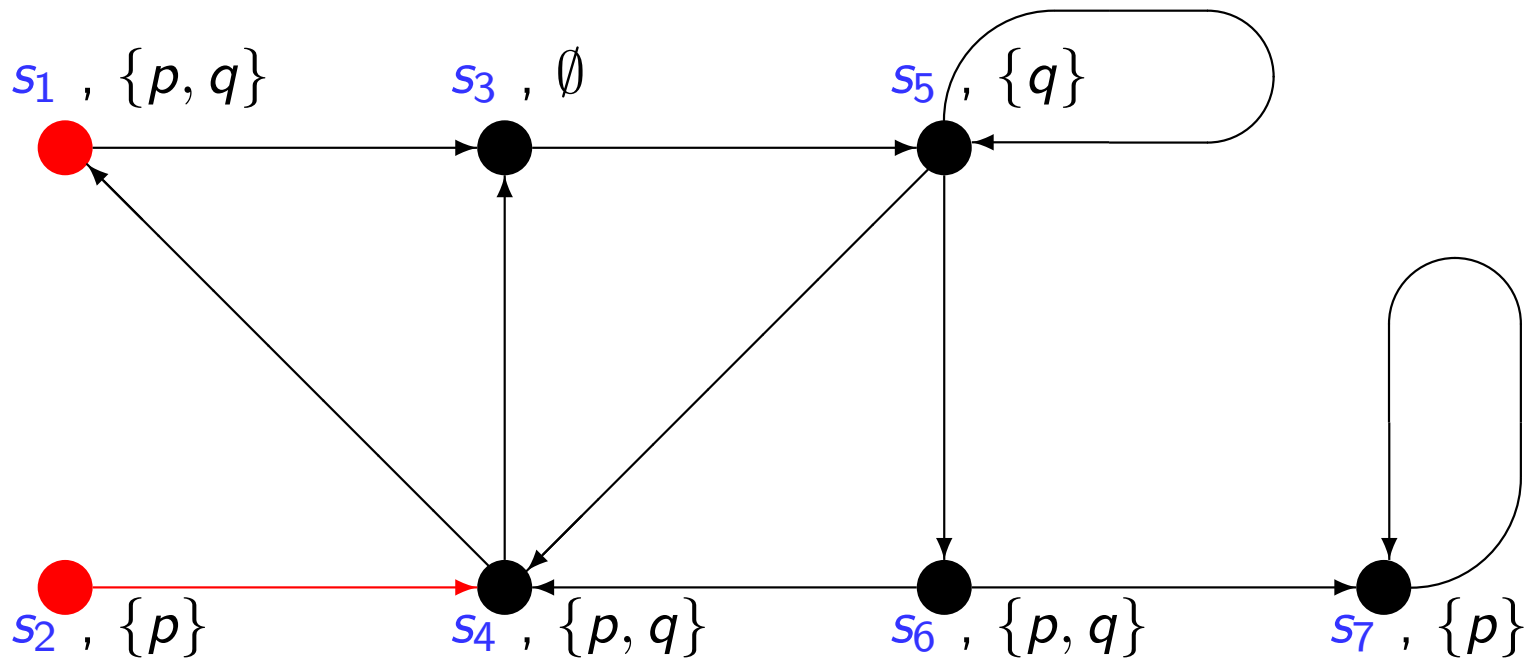
$p = \textbf{true}$
$q = \textbf{false}$



РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

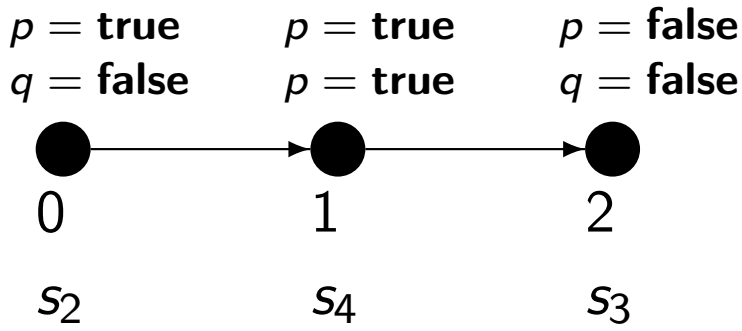
Пример LTS

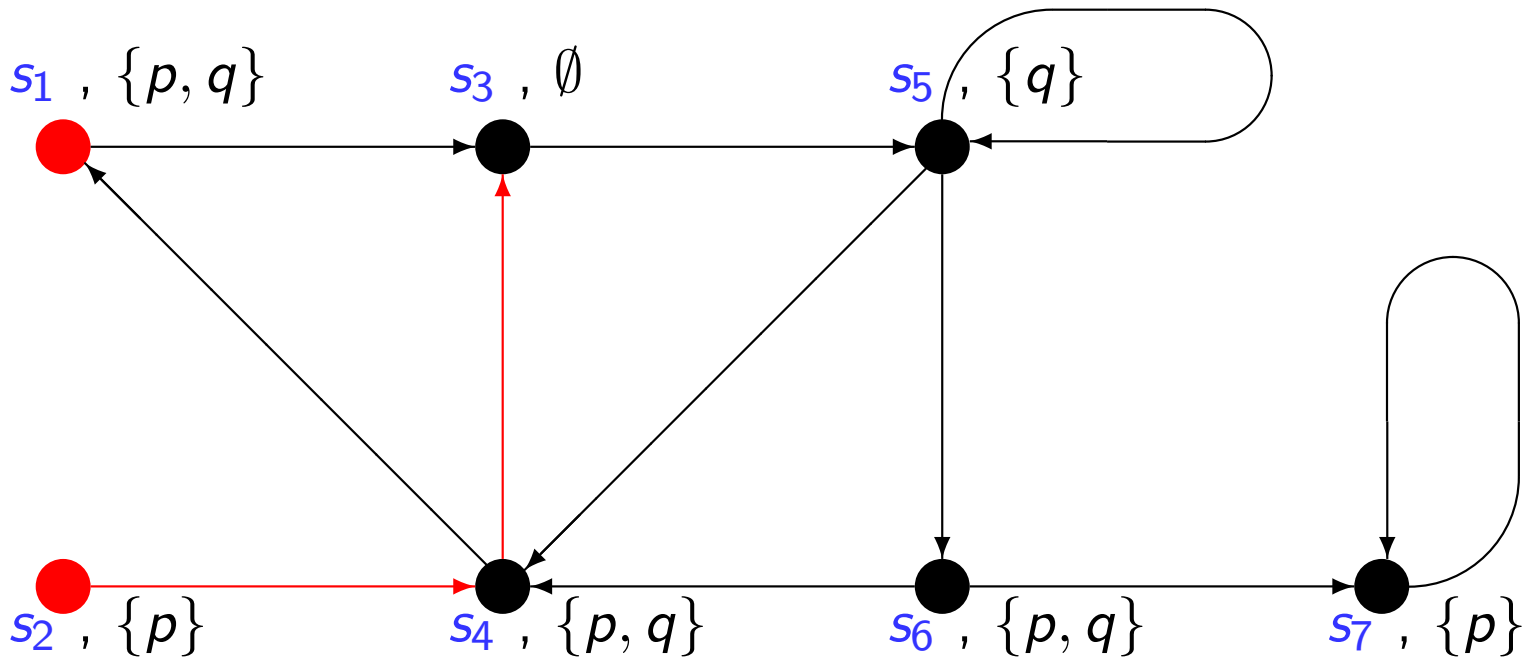




РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

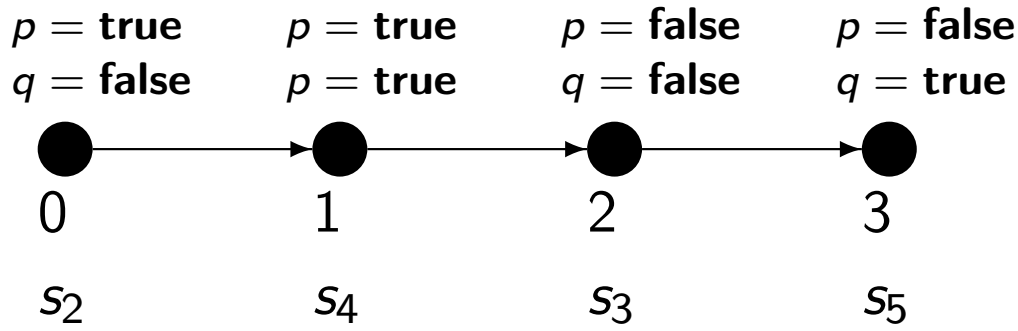
Пример LTS

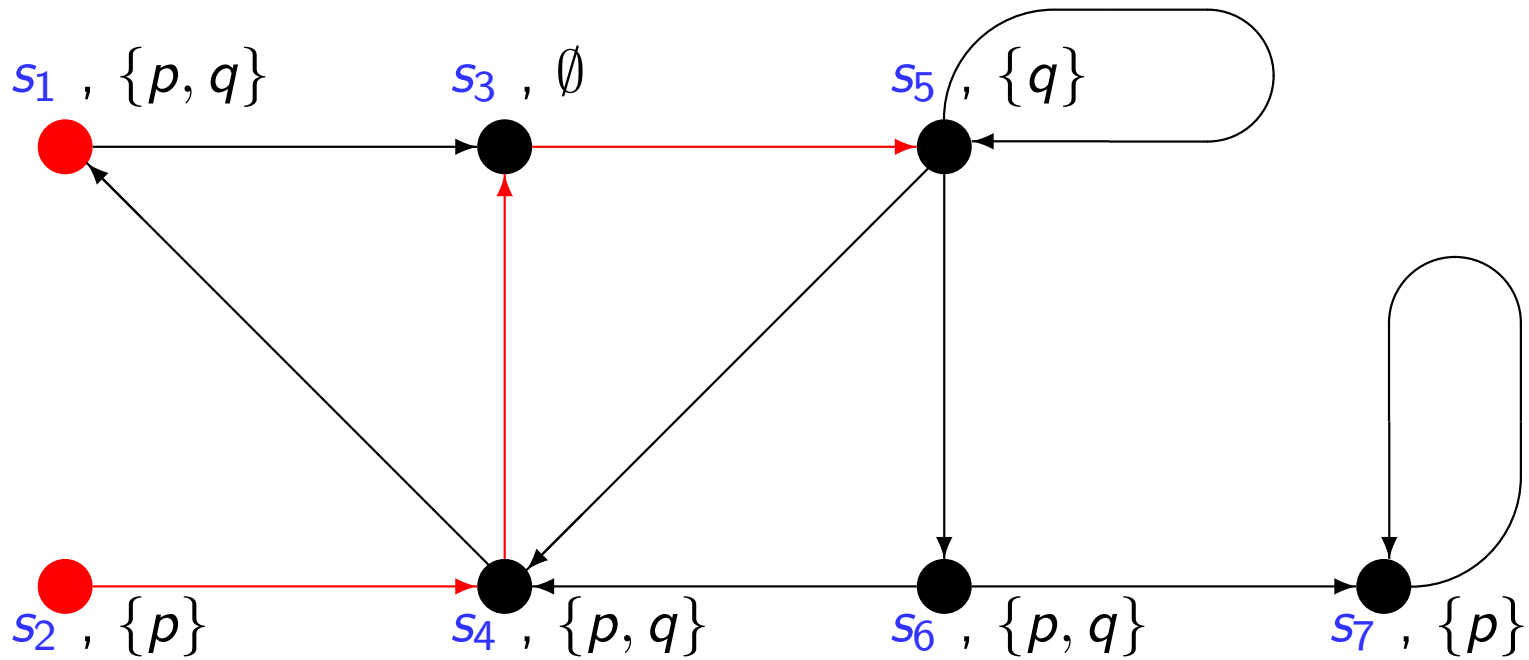




РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

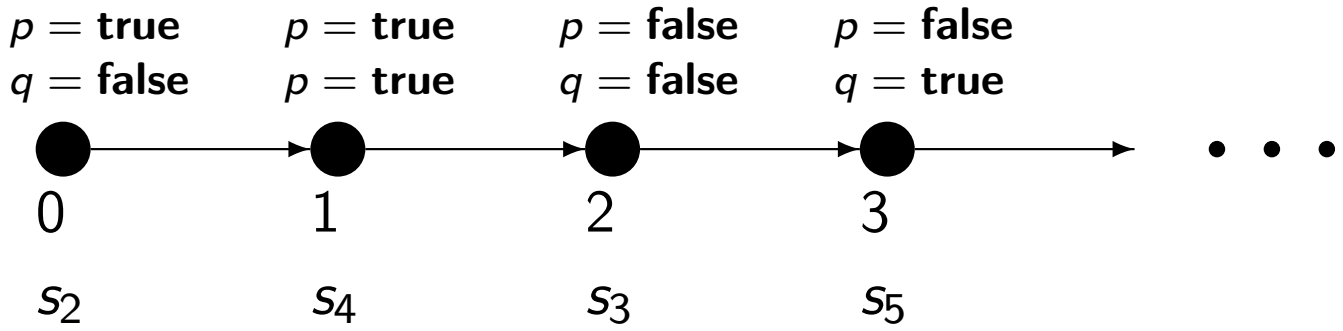
Пример LTS

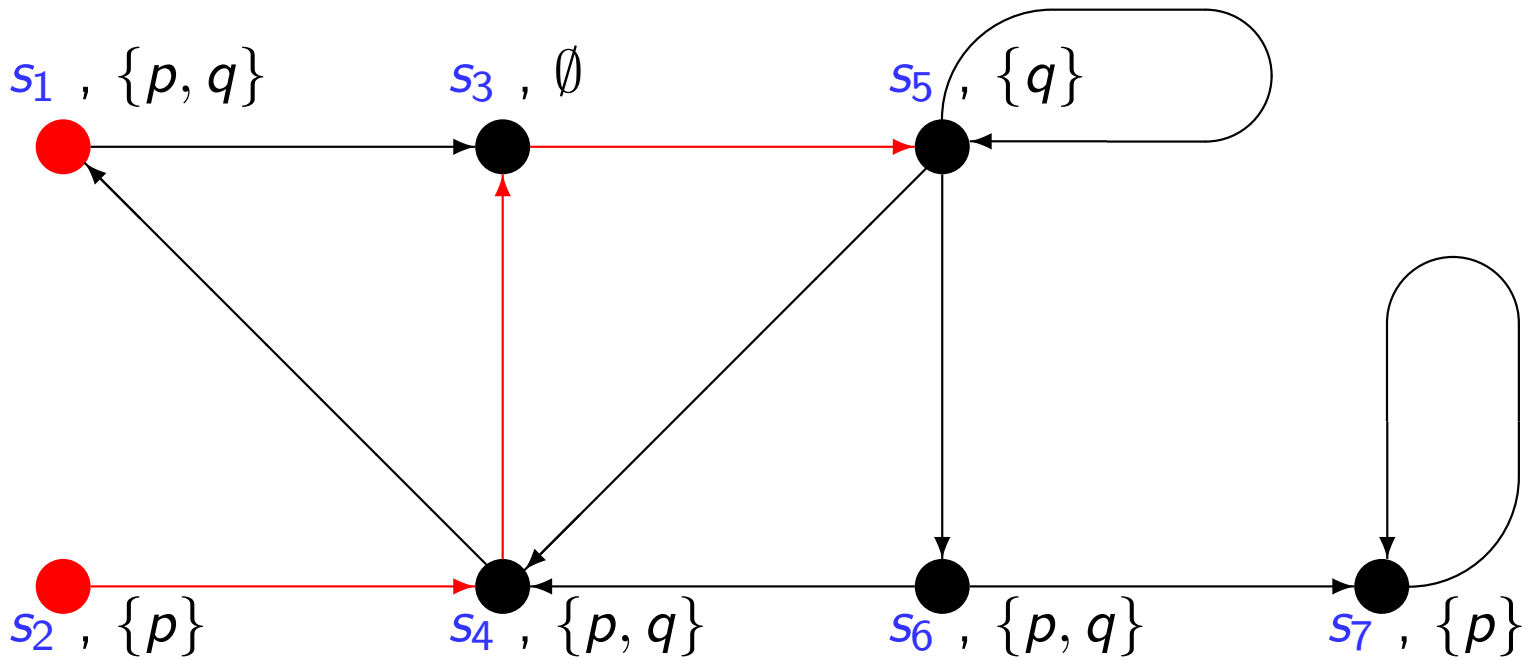




РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

Пример LTS





РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS и распределенные программы

LTS, соответствующую программе π, можно построить так:

- Состояниями LTS полагаются состояния вычисления программы. Состояние вычисления программы π — это пара (состояние управления, состояние данных). Состояние управления — это значение счетчика команд программы. Состояние данных — это подстановка, указывающая соответствие между переменными и их значениями.
- Если в точке $count_1$ программы π выполняется оператор op, преобразующий данные из состояния ξ_1 в состояние ξ_2 и передающий управление в точку $count_2$, то в LTS имеется переход $(count_1, \xi_1) \longrightarrow (count_2, \xi_2)$.
- Атомарным высказыванием p может быть любая формула логики предикатов, зависящая от переменных программы и счетчика команд. Разметка ρ определяется так: $p \in \rho((count, \xi)) \iff I_\pi \models p[(count, \xi)]$.

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

Программа драйвера π_1

while true do

L_1: **while** $R \neq$ *free* **do wait od** ;

L_2: R = *busy*;

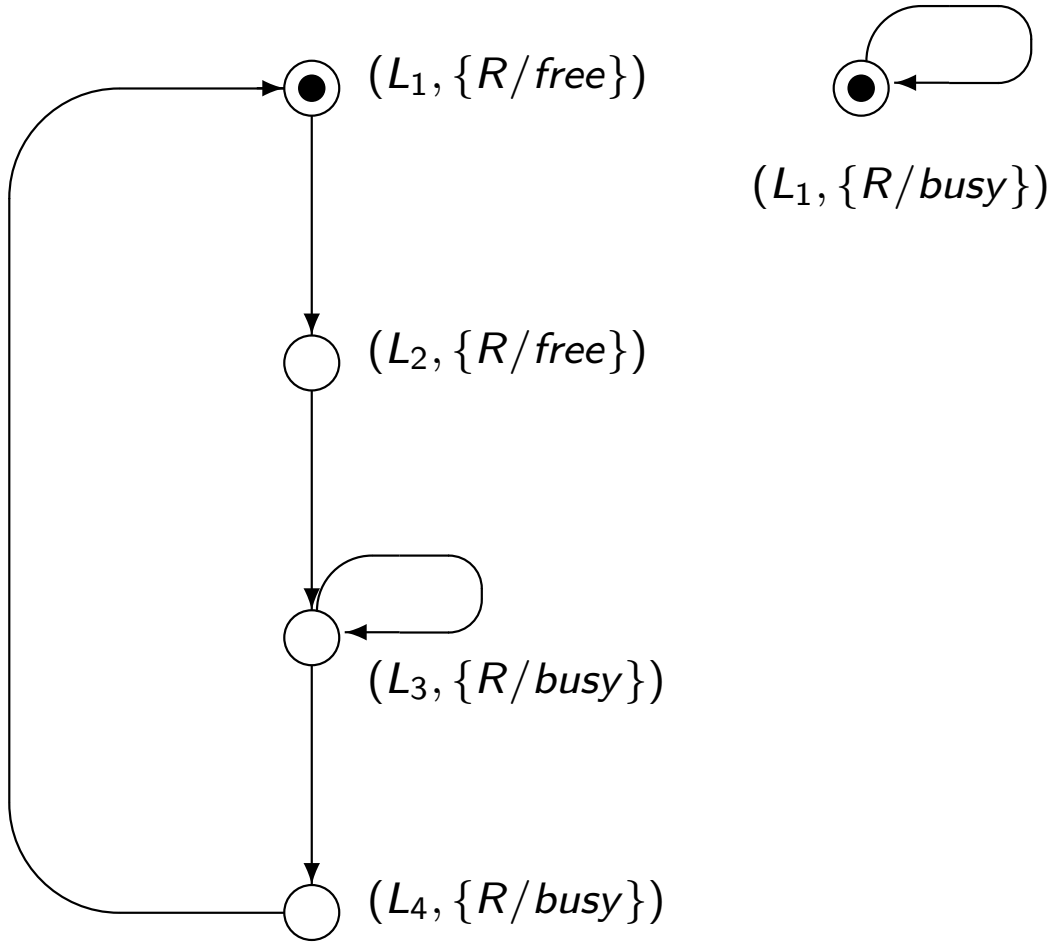
L_3: **output**(X,*printer*);

L_4: R = *free*;

od

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS для программы π_1



РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS и распределенные программы

Некоторые переменные программы могут быть доступны для других программ (т. н. разделяемые переменные) или для окружающей среды (датчики, сенсоры, средства управления).

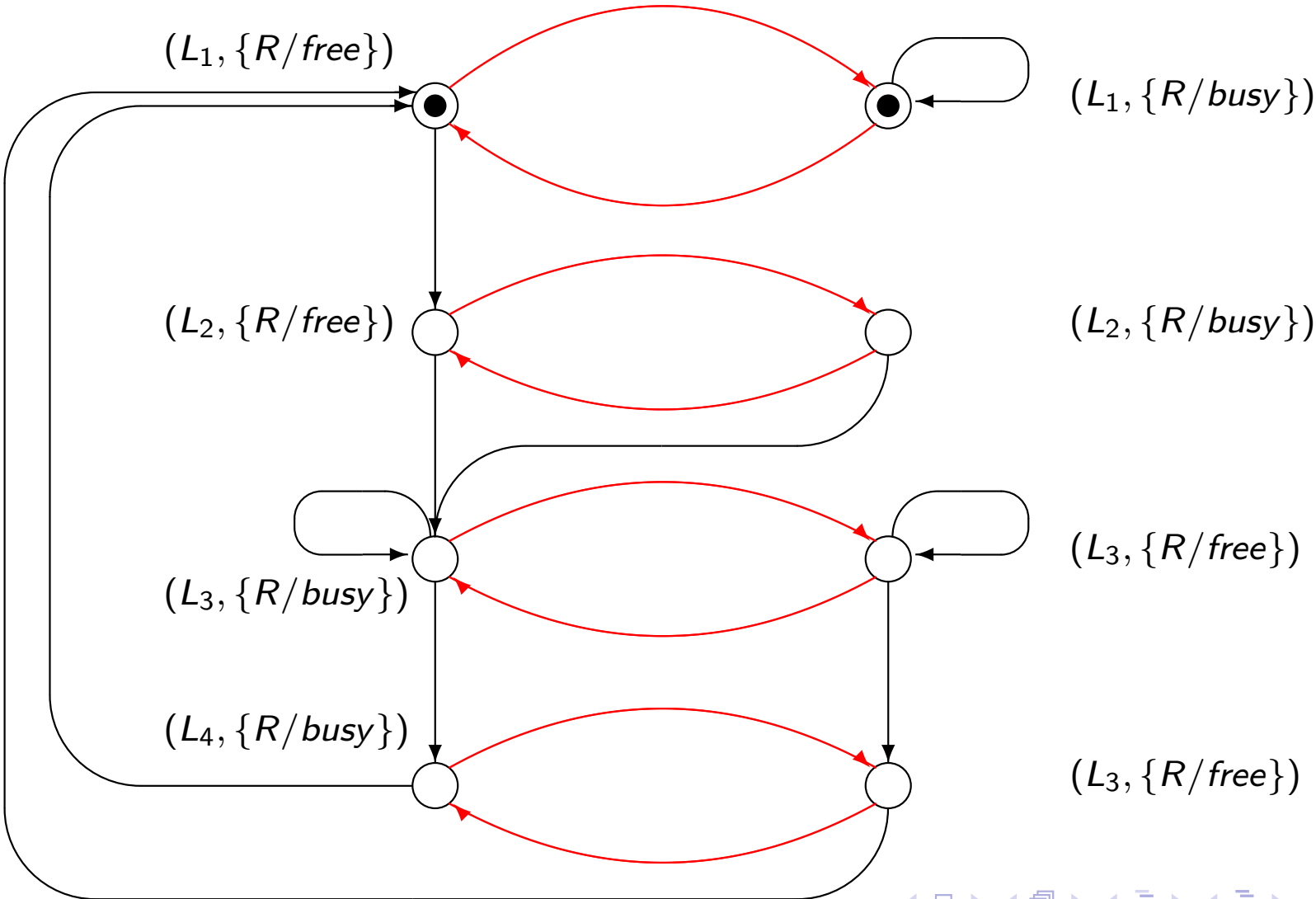
В том случае, когда программа π взаимодействует с окружающей средой, в LTS M_π вносятся следующие изменения:

для каждого состояния (l, ξ) вводятся переходы $(l, \xi) \longrightarrow (l, \xi')$ во всевозможные состояния (l, ξ'), в которых подстановки ξ', отличающиеся от ξ только значениями переменных, доступных для окружающей среды.

Например, полагая, что регистр принтера R — это тумблер, который может переключаться сколь угодно часто в разные моменты времени, получим следующую LTS, описывающую взаимодействие драйвера принтера с окружающей средой (пользователем принтера).

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS для системы $\pi_1 \parallel \mathit{environment}$



РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS и распределенные программы

LTS для распределенной системы, состоящей из двух процессов π_1 и π_2, взаимодействующих посредством разделяемых переменных, строится на основе семантики чередующихся вычислений.

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS и распределенные программы

LTS для распределенной системы, состоящей из двух процессов π_1 и π_2, взаимодействующих посредством разделяемых переменных, строится на основе семантики чередующихся вычислений.

Состояниями LTS для системы $\pi_1 \parallel \pi_2$ объявляются наборы $(count_1, count_2, \xi_1, \xi_2, \chi)$, где

- $count_1, count_2$ — значения счетчиков команд процессов π_1 и π_2,
- ξ_1, ξ_2 — подстановки, определяющие значения локальных переменных процессов π_1 и π_2,
- χ — подстановка, определяющая значения разделяемых переменных.

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS и распределенные программы

LTS для распределенной системы, состоящей из двух процессов π_1 и π_2, взаимодействующих посредством разделяемых переменных, строится на основе семантики чередующихся вычислений.

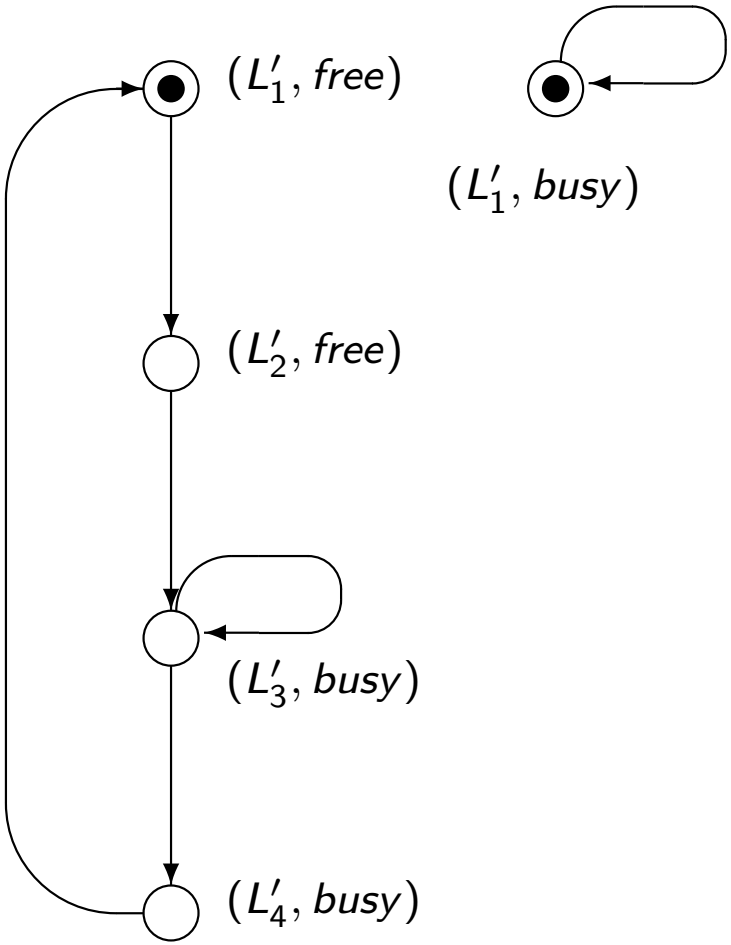
Переход $(count_1, count_2, \xi_1, \xi_2, \chi) \longrightarrow (count_1', count_2', \xi_1', \xi_2', \chi')$ возможен в том и только том случае, когда выполнено одно из двух условий:

- в LTS $M(\pi_1)$ есть переход $(count_1, \xi_1, \chi) \longrightarrow (count_1', \xi_1', \chi')$ и при этом $count_2' = count_2$, $\xi_2' = \xi_2$;
- в LTS $M(\pi_2)$ есть переход $(count_2, \xi_2, \chi) \longrightarrow (count_2', \xi_2', \chi')$ и при этом $count_1' = count_1$, $\xi_1' = \xi_1$.

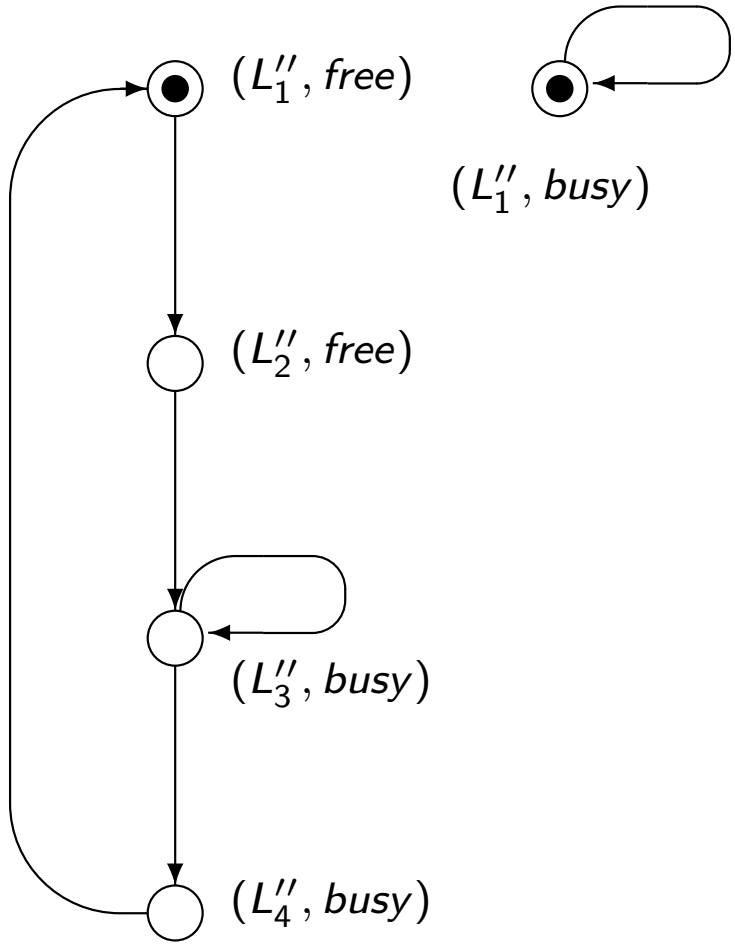
Таким образом, в семантике чередующихся вычислений параллельное выполнение процессов распределенной системы моделируется недетерминированным выбором того или иного порядка, в котором выполняются действия разных процессов.

РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS для программы π'

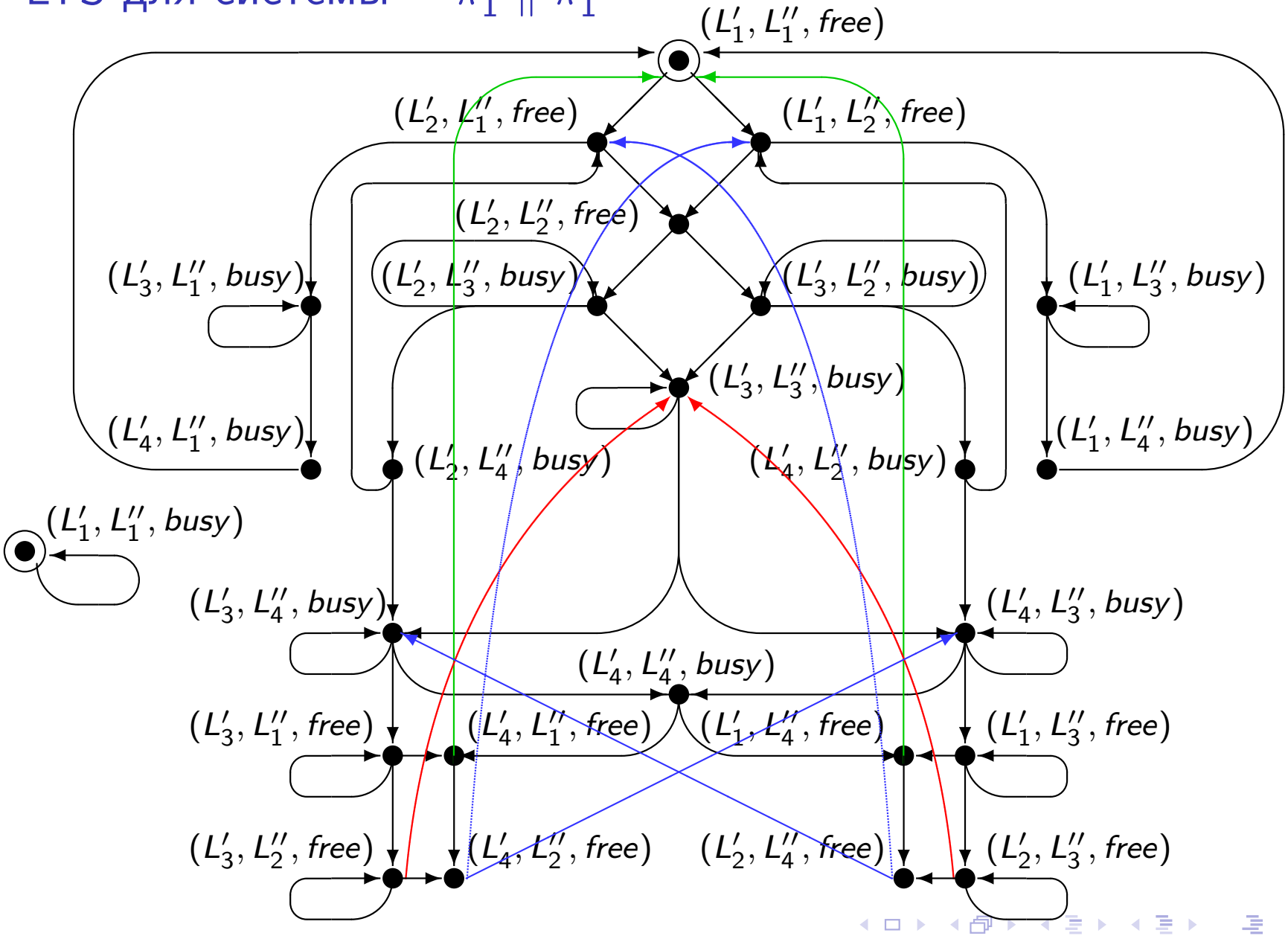


и программы π''



РАЗМЕЧЕННЫЕ СИСТЕМЫ ПЕРЕХОДОВ

LTS для системы $\pi'_1 \parallel \pi''_1$



ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Итак, для верификации распределенных программ нужны

- требования правильности (спецификации) вычислений программы, представленные формулой PLTL φ,
- математическая модель распределенной программы π, представленная конечной LTS $M(\pi)$.

Тогда для проверки правильности программы π достаточно проверить, что для любой трассы tr, $tr \in Tr_0(M(\pi))$, формула φ выполняется в темпоральной интерпретации $I(tr)$, т. е. имеет место $I(tr), 0 \models \varphi$.

Воспользуемся записью $M \models \varphi$ для обозначения утверждения «для любой трассы tr, $tr \in Tr_0(M)$, имеет место $I(tr), 0 \models \varphi$».

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Задача верификации моделей программ (model checking) для PLTL формулируется так:

для заданной формулы PLTL φ и LTS M проверить $M \models \varphi$.

Существует ли алгоритм решения задачи верификации моделей программ?

КОНЕЦ ЛЕКЦИИ 21.

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 22.

Задача верификации моделей программ. Подформулы Фишера-Ладнера. Табличный метод верификации моделей программ. Алгоритм верификации моделей программ.

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Задача model checking для PLTL

Для заданной формулы PLTL φ и конечной LTS M проверить $M \models \varphi$.

Почему задача model checking непроста? Потому что

- выполнимость формул PLTL проверяется на бесконечных интерпретациях,
- В LTS M имеется бесконечно много интерпретаций (трасс).

Почему задача model checking имеет эффективное решение? Потому что

- все это бесконечное множество бесконечных интерпретаций «упаковано» в **конечную** структуру — LTS M.

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Замысел табличного метода

1. Вместо проверки выполнимости φ во всех интерпретациях лучше заняться поиском контрмодели — интерпретации I, в которой не выполняется φ.
2. Выполнимость всякой формулы ψ полностью определяется выполнимостью ее подформул. Поэтому (не)выполнимость формул можно проверять индуктивно.
3. (Не)выполнимость формулы на одной из трасс LTS M, начинающейся в состоянии s, — это свойство состояния s. Значит, проверяя (не)выполнимость всех подформул формулы φ для всех состояний LTS M, можно вычислить множество $\overline{S}_\varphi$ всех тех состояний, в которых не выполняется формула φ. Если $S_0 \cap \overline{S}_\varphi \neq \varnothing$, то $M \not\models \varphi$.

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Вспомогательные определения и обозначения

Для заданной LTS $M = \langle \mathcal{AP}, S, S_0, \longrightarrow, \rho \rangle$, трассы $tr = s_{i_0}, s_{i_1}, \ldots, s_{i_n}, s_{i_{n+1}}, \ldots$ в LTS M и формулы PLTL φ будем использовать запись

- $tr \models \varphi$ для обозначения отношения выполнимости $I(tr), 0 \models \varphi$;
- $tr[j]$ для обозначения j-го состояния s_{i_j} в трассе tr;
- $tr|_j$ для обозначения трассы $tr' = s_{i_j}, s_{i_{j+1}}, \ldots$, являющейся суффиксом трассы tr, начинающейся состоянием s_{i_j}.

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Утверждение 1.

Для любой LTS M и формула PLTL φ верно

$$M \not\models \varphi \iff \text{существует такая начальная трасса } tr,\ tr \in Tr_0(M), \text{ для которой } tr \not\models \varphi.$$

Доказательство.

Самостоятельно.

Таким образом, вместо задачи $M \models \varphi$ мы будем рассматривать другую задачу:

найти в LTS M начальную трассу tr, для которой $tr \not\models \varphi$.

Если такой трассы найти не удастся, то верно $M \models \varphi$.

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Приведение формулы к позитивной форме

Применяя равносильные преобразования, упростим формулу φ.

Этап 1. Удаление импликации $\rightarrow$ и темпоральных операторов $\mathbf{G}, \mathbf{F}$ на основании законов взаимной зависимости

$$\models \psi \rightarrow \psi \equiv \neg\psi \vee \chi;$$

$$\models \mathbf{F}\psi \equiv \mathbf{true}\ \mathbf{U}\psi; \qquad \models \mathbf{G}\psi \equiv \mathbf{false}\ \mathbf{R}\psi.$$

Этап 2. Продвижение $\neg$ вглубь формулы на основании законов двойственности

$$\models \neg(\psi\&\chi) \equiv \neg\psi \vee \neg\chi; \qquad \models \neg(\psi \vee \chi) \equiv \neg\psi\&\neg\chi;$$

$$\models \neg\neg\psi \equiv \psi; \qquad \models \neg\mathbf{X}\psi \equiv \mathbf{X}\neg\psi;$$

$$\models \neg(\psi\mathbf{U}\chi) \equiv \neg\psi\mathbf{R}\neg\chi; \qquad \models \neg(\psi\mathbf{R}\chi) \equiv \neg\psi\mathbf{U}\neg\chi.$$

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Утверждение 2.

В результате применения равносильных преобразований этапов 1 и 2 любая формула PLTL φ приводится к равносильной формуле φ', представленной в позитивной форме, в которой

- используются только логические связки $\vee, \&, \neg$ и темпоральные операторы $\mathbf{X}, \mathbf{F}, \mathbf{G}$,
- связка $\neg$ применяется только к атомарным высказываниям $p,\ p \in \mathcal{AP}$.

Доказательство.

Самостоятельно.

ЗАДАЧА ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Приведение формулы к позитивной форме

Пример.

$\varphi = \mathbf{G}(\mathit{free}\ \&\ \mathbf{X}\mathit{busy}\ \rightarrow\ \mathbf{XF}(\mathit{pr}_1\ \vee\ \mathit{pr}_2))$.

Этап 1.

$\varphi' = \mathbf{false}\ \mathbf{R}\ (\neg(\mathit{free}\ \&\ \mathbf{X}\mathit{busy})\ \vee\ \mathbf{X}(\mathbf{true}\ \mathbf{U}(\mathit{pr}_1\ \vee\ \mathit{pr}_2)))$.

Этап 2.

$\varphi_1 = \mathbf{false}\ \mathbf{R}\ (\neg\mathit{free}\ \vee\ \mathbf{X}\neg\mathit{busy}\ \vee\ \mathbf{X}(\mathbf{true}\ \mathbf{U}(\mathit{pr}_1\ \vee\ \mathit{pr}_2)))$.

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Пусть φ_1 — формула PLTL в позитивной форме. Тогда множеством подформул Фишера–Ладнера называется наименьшее множество формул PLTL $FLSub_{\varphi_1}$, содержащее формулу φ_1 и удовлетворяющее следующим условиям:

- если $p \in FLSub_{\varphi_1}$ и $p \in \mathcal{AP}$, то $\neg p \in FLSub_{\varphi_1}$,
- если $\psi \& \chi \in FLSub_{\varphi_1}$, то $\{\psi, \chi\} \subseteq FLSub_{\varphi_1}$,
- если $\psi \vee \chi \in FLSub_{\varphi_1}$, то $\{\psi, \chi\} \subseteq FLSub_{\varphi_1}$,
- если $\neg\psi \in FLSub_{\varphi_1}$, то $\psi \in FLSub_{\varphi_1}$,
- если $\mathbf{X}\psi \in FLSub_{\varphi_1}$, то $\psi \in FLSub_{\varphi_1}$,
- если $\psi \mathbf{U} \chi \in FLSub_{\varphi_1}$, то $\{\psi, \chi, \mathbf{X}(\psi \mathbf{U} \chi)\} \subseteq FLSub_{\varphi_1}$,
- если $\psi \mathbf{R} \chi \in FLSub_{\varphi_1}$, то $\{\psi, \chi, \mathbf{X}(\psi \mathbf{R} \chi)\} \subseteq FLSub_{\varphi_1}$.

Утверждение 3.

Если φ_1 содержит n логических связок и темпоральных операторов, то $|FLSub_{\varphi_1}| \leq 3n$.

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Пример.

Пусть

$\varphi_1 = \mathbf{false}\ \mathbf{R}\ (\neg \mathit{free} \vee \mathbf{X}\neg \mathit{busy} \vee \mathbf{X}(\mathbf{true}\ \mathbf{U}(\mathit{pr}_1 \vee \mathit{pr}_2)))$.

Тогда

$$
\begin{aligned}
\mathit{FLSub}_{\varphi_1} = \{&\varphi_1,\\
&\mathbf{false}, \mathbf{X}\varphi_1,\ \neg \mathit{free} \vee \mathbf{X}\neg \mathit{busy} \vee \mathbf{X}(\mathbf{true}\ \mathbf{U}(\mathit{pr}_1 \vee \mathit{pr}_2)),\\
&\neg \mathit{free},\ \mathbf{X}\neg \mathit{busy},\ \mathbf{X}(\mathbf{true}\ \mathbf{U}(\mathit{pr}_1 \vee \mathit{pr}_2)),\\
&\mathit{free}, \neg \mathit{busy},\ \mathbf{true}\ \mathbf{U}(\mathit{pr}_1 \vee \mathit{pr}_2),\\
&\mathit{busy}, \mathbf{true}, \mathit{pr}_1 \vee \mathit{pr}_2,\\
&\mathit{pr}_1,\ \mathit{pr}_2,\ \neg \mathit{pr}_1,\ \neg \mathit{pr}_2\}.
\end{aligned}
$$

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Next-подформулы

Пусть φ_1 — формула PLTL в позитивной форме и $FLSub_{\varphi_1}$ — множеством подформул Фишера–Ладнера формулы φ_1.

Тогда запись $XSub_{\varphi_1}$ будет обозначать множество всех тех подформул Фишера–Ладнера, которые начинаются оператором **X** (neXttime), т. е.

$$XSub_{\varphi_1} = \{\psi \ : \ \psi = \mathbf{X}\chi, \ \psi \in FLSub_{\varphi_1}\}.$$

Пример.

Пусть

$\varphi_1 = \mathbf{false}\ \mathbf{R}\ (\neg free \ \vee \ \mathbf{X}\neg busy \ \vee \ \mathbf{X}(\mathbf{true}\ \mathbf{U}(pr_1 \ \vee \ pr_2)))$.

Тогда

$$\begin{aligned} XSub_{\varphi_1} \ = \ & \{\mathbf{X}\varphi_1, \ \mathbf{X}\neg busy, \\ & \mathbf{X}(\mathbf{true}\ \mathbf{U}(pr_1 \ \vee \ pr_2))\}. \end{aligned}$$

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

(Until-Release)-подформулы

Пусть φ_1 — формула PLTL в позитивной форме и $FLSub_{\varphi_1}$ — множеством подформул Фишера–Ладнера формулы φ_1.

Тогда запись $URSub_{\varphi_1}$ будет обозначать множество всех тех подформул Фишера–Ладнера, которые начинаются оператором **U** (Until) или **R** (Release), т. е.

$$\begin{aligned} USub_{\varphi_1} \quad = \quad & \{\psi \ : \ \psi = \chi_1 \mathbf{U} \chi_2, \ \psi \in FLSub_{\varphi_1}\} \cup \\ & \{\psi \ : \ \psi = \chi_1 \mathbf{R} \chi_2, \ \psi \in FLSub_{\varphi_1}\}. \end{aligned}$$

Пример.

Пусть

$\varphi_1 = \mathbf{false}\ \mathbf{R}\ (\neg free \ \vee \ \mathbf{X} \neg busy \ \vee \ \mathbf{X}(\mathbf{true}\ \mathbf{U}(pr_1 \ \vee \ pr_2)))$.

Тогда

$$USub_{\varphi_1} \quad = \quad \{\varphi_1, \mathbf{true}\ \mathbf{U}(pr_1 \ \vee \ pr_2)\}.$$

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Согласованные множества подформул

Пусть φ_1 — формула PLTL в позитивной форме, и $FLSub_{\varphi_1}$ — множество подформул Фишера–Ладнера для φ_1.
Тогда согласованным множеством подформул формулы φ_1 называется всякое подмножество B, $B \subseteq FLSub_{\varphi_1}$, удовлетворяющее следующим условиям:

1. $\mathbf{true} \in B$, $\mathbf{false} \notin B$,
2. для любого атомарного высказывания p, $p \in \mathcal{AP} \cap FLSub_{\varphi_1}$, выполняется **в точности одно из двух** включений: либо $p \in B$, либо $\neg p \in B$;
3. $\psi \vee \chi \in B \iff \psi \in B$ или $\chi \in B$,
4. $\psi \& \chi \in B \iff \psi \in B$ и $\chi \in B$,
5. $\psi \mathbf{U} \chi \in B \iff \chi \in B$ или $\{\psi, \mathbf{X}(\psi \mathbf{U} \chi)\} \subseteq B$,
6. $\psi \mathbf{R} \chi \in B \iff \chi \in B$ и при этом $\psi \in B$ или $\mathbf{X}(\psi \mathbf{R} \chi) \in B$.

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Согласованные множества подформул

Согласованные множества подформул — это максимальные множества формул, которые не содержат «явных» противоречий, т. е. таких противоречий, которые можно обнаружить в текущий момент времени.

Например, множество, состоящее из двух формул

$\mathbf{X}p$ — завтра я пойду на лекцию,

$\mathbf{X}\neg p$ — завтра я не пойду на лекцию,

может быть согласованным (хотя и противоречивым), поскольку сегодня возможное противоречие, содержащееся в этих высказываниях, не проявляется.

Согласованное множество подформул является аналогом семантической таблицы — оно выражает наше пожелание сделать все утверждения, содержащиеся в этом множестве, истинными, а все утверждения, не содержащиеся в нем, — ложными.

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Согласованные множества подформул

Пример.

Пусть

$$
\begin{aligned}
FLSub_{\varphi_1} \quad = \quad & \{free,\ busy,\ pr_1,\ pr_2,\ \neg free,\ \neg busy,\ \neg pr_1,\ \neg pr_2,\\
& pr_1 \vee pr_2,\\
& \mathbf{true}\ \mathbf{U}(pr_1 \vee pr_2),\\
& \mathbf{X}\neg busy, \mathbf{X}(\mathbf{true}\ \mathbf{U}(pr_1 \vee pr_2)),\\
& \neg free \vee \mathbf{X}\neg busy \vee \mathbf{X}(\mathbf{true}\ \mathbf{U}(pr_1 \vee pr_2)),\\
& \varphi_1, \mathbf{X}\varphi_1\}.
\end{aligned}
$$

Тогда одним из согласованных множеств подформул формулы φ_1 является множество

$$
\begin{aligned}
B \quad = \quad & \{\mathbf{true}, pr_1,\ \neg pr_2, \neg free,\ busy, \mathbf{X}\neg busy,\\
& \mathbf{true}\ \mathbf{U}(pr_1 \vee pr_2),\ \mathbf{X}(\mathbf{true}\ \mathbf{U}(pr_1 \vee pr_2)), \varphi_1\}.
\end{aligned}
$$

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Утверждение 4.

Пусть I — произвольная темпоральная интерпретация, и φ_1 — произвольная формула в позитивной форме.
Тогда для любого момента времени n множество формул

$$B_n = \{\psi \ : \ \psi \in FLSub_{\varphi_1} \text{ и } I, n \models \psi\}$$

является согласованным.

Доказательство.

Самостоятельно. Непосредственно из определения согласованного множества.

А верно ли обратное утверждение: каждое согласованное множество формул выполнимо в некоторой интерпретации в начальный момент времени?

ПОДФОРМУЛЫ ФИШЕРА–ЛАДНЕРА

Утверждение 5.

Пусть φ_1 — формула PLTL в позитивной форме. Тогда

1. для любой пары $B' \subseteq \mathcal{AP} \cap FLSub_{\varphi_1}$, $B'' \subseteq XSub_{\varphi_1}$, существует такое согласованное множество подформул B, для которого верно $B \cap \mathcal{AP} = B'$, $B \cap XSub_{\varphi_1} = B''$;
2. для любой пары B_1 и B_2 согласованных множеств подформул Фишера-Ладнера φ_1 верны соотношения $B_1 = B_2 \iff B_1 \cap \mathcal{AP} = B_2 \cap \mathcal{AP}$ и $B_1 \cap XSub_{\varphi_1} = B_1 \cap XSub_{\varphi_1}$.

Доказательство.

Самостоятельно.

Утверждение 6.

Если φ_1 содержит n логических связок и темпоральных операторов, то число различных согласованных множеств подформул Фишера-Ладнера не превосходит величины 2^{3n}.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Пусть задана формулы PLTL φ и конечная LTS $M = \langle \mathcal{AP}, S, S_0, \longrightarrow, \rho \rangle$.
Нужно проверить выполнимость $M \models \varphi$.

Для этого

1. формула φ приводится к позитивной форме φ_1,
2. для формулы φ_1 строятся
   - множество подформул Фишера–Ладнера $FLSub_{\varphi_1}$,
   - множество Next-подформул $XSub_{\varphi_1}$,
   - множество U-подформул $FLSub_{\varphi_1}$,
   - совокупность Con_{φ_1} всех возможных согласованных множеств подформул Фишера–Ладнера.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Системой Хинтикки для формулы PLTL φ_1 и LTS M называется раскрашенный ориентированный граф $G_{\varphi_1,M} = (V, E)$ с множеством вершин V и множеством дуг E, которые устроены так:

$$V = \{(s, B) \ : \ s \in S,\ B \in Con_{\varphi_1},\ \rho(s) = B \cap \mathcal{AP}\},$$

т. е. вершинами графа являются всевозможные пары (состояние s, согласованное множество B), для которых разметка $\rho(s)$ состояния s подтверждает истинность всех атомарных высказываний множества B;

$$\begin{aligned} E \ = \ \{\langle (s', B'), (s'', B'') \rangle \ : \ & s' \longrightarrow s'' \\ & \text{и для любой Next-подформулы } \mathbf{X}\psi, \mathbf{X}\psi \in XSub_{\varphi_1}, \\ & \text{верно } \mathbf{X}\psi \in B' \iff \psi \in B''\}, \end{aligned}$$

т. е. дугами графа являются все такие переходы LTS M, которые позволяют подтвердить все обещания $\mathbf{X}\psi$ выполнить ψ в следующий момент времени.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Теперь проведем раскраску вершин графа $\Gamma_{\varphi_1,M} = (V, E)$.
Рассмотрим множество (Until-Release)-подформул
$URSub_{\varphi_1} = \{\chi'_1\mathbf{U}\chi''_1, \ldots, \chi'_k\mathbf{U}\chi''_k, \chi'_{k+1}\mathbf{R}\chi''_{k+1}, \ldots, \chi'_{k+m}\mathbf{R}\chi''_{k+m}\}$.
Каждой формуле ψ_i из множества $URSub_{\varphi_1}$ сопоставим индивидуальный цвет i.
Раскрасим в цвет i все вершины (s, B), для которых выполнено **хотя бы одно** из двух условий

в случае, когда $\psi_i = \chi'_i\mathbf{U}\chi''_i$:	в случае, когда $\psi_i = \chi'_i\mathbf{R}\chi''_i$:
1) $\chi''_i \in B$, 2) $\mathbf{X}(\chi'_i\mathbf{U}\chi''_i) \notin B$.	1) $\chi''_i \notin B$, 2) $\mathbf{X}(\chi'_i\mathbf{R}\chi''_i) \in B$.

Бесконечный маршрут

$$(s_{i_1}, B_{i_1}), (s_{i_2}, B_{i_2}), \ldots, (s_{i_n}, B_{i_n}), \ldots$$

в графе $\Gamma_{\varphi_1,M}$ назовем радужным, если в нем бесконечно часто встречаются вершины каждого цвета $1, 2, \ldots, k$.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Основная теорема

Для любой формулы PLTL φ_1 в позитивной форме и LTS $M = \langle \mathcal{AP}, S, S_0, \longrightarrow, \rho \rangle$

$$M \not\models \varphi_1$$

$$\Updownarrow$$

в графе $\Gamma_{\varphi_1, M}$ существует хотя бы один радужный маршрут, исходящий из вершины $v_0 = (s_0, B_0)$, в которой $s_0 \in S_0$ и $\varphi_1 \notin B_0$.

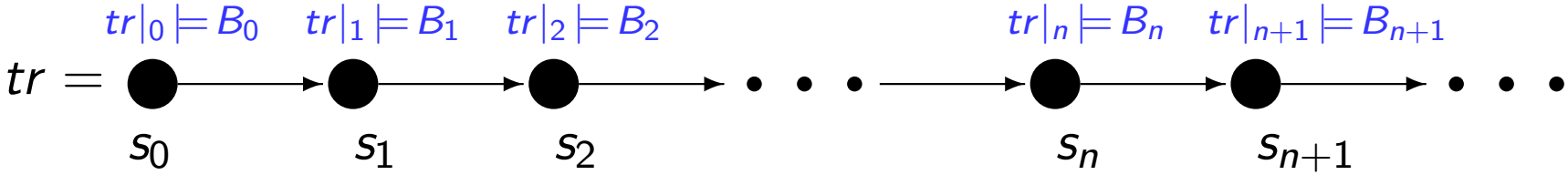
ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Доказательство.

($\Uparrow$) Предположим, что в графе $\Gamma_{\varphi_1,M}$ есть радужный маршрут

$$(s_0, B_0), (s_1, B_1), \ldots, (s_n, B_n), (s_{n+1}, B_{n+1}), \ldots$$

указанного вида, в котором $\varphi_1 \notin B_0$.
Тогда согласно определению системы Хинтикки $\Gamma_{\varphi_1,M}$ в LTS M есть начальная трасса



Покажем, что для любой формулы ψ, $\psi \in FLSub_{\varphi_1}$, и для любого $n, n \geq 0$, верно

$$tr|_n \models \psi \iff \psi \in B_n \ .$$

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Доказательство.

Если удастся показать, что

$$tr|_n \models \psi \iff \psi \in B_n \qquad (*)$$

то, учитывая $\varphi_1 \notin B_0$, придем к заключению $tr \not\models \varphi_1$.
Для доказательства соотношения $(*)$ воспользуемся индукцией по числу связок в формуле ψ.

Базис индукции. $p \in \mathcal{AP}$.

$p \in B_n \iff p \in \xi(s_n) \iff tr|_n \models p$.

$\neg p \in B_n \iff p \notin B_n \iff p \notin \xi(s_n) \iff tr|_n \not\models p \iff tr|_n \models \neg p.$

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Доказательство.

Индуктивный переход.

1. Логические связки & и $\vee$.

$\psi_1 \& \psi_2 \in B_n \iff \psi_1 \in B_n$ и $\psi_2 \in B_n \iff tr|_n \models \psi_1$ и $tr|_n \models \psi_1$
$\iff tr|_n \models \psi_1 \& \psi_1$.

2. Темпоральный оператор $\mathbf{X}$.

$\mathbf{X}\psi \in B_n \iff \psi \in B_{n+1} \iff tr|_{n+1} \models \psi \in B_{n+1} \iff tr|_n \models \mathbf{X}\psi.$

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Индуктивный переход.

3. Темпоральный оператор $\mathbf{R}$.

3.1. Покажем $\psi_1 \mathbf{R} \psi_2 \in B_n \Longrightarrow tr|_n \models \psi_1 \mathbf{R} \psi_2$.

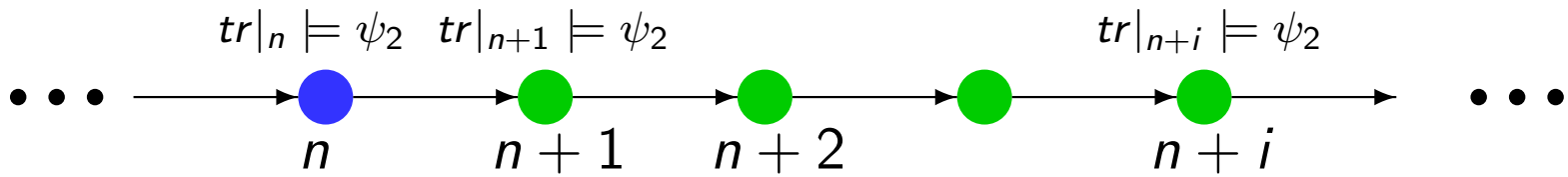
Заметим, что согласно определению согласованного множества $\psi_1 \mathbf{R} \psi_2 \in B \iff \psi_2 \in B$ и при этом $\psi_1 \in B$ или $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \in B$.

Пусть $\psi_1 \mathbf{R} \psi_2 \in B_n$. Тогда возможны 2 случая.

Вариант 1. $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \in B_{n+i}$ для любого i, $i \geq 0$.

Тогда по определению $\Gamma_{\varphi_1, M}$ в каждом множестве B_{n+i}, $i \geq 0$, содержится формула $\psi_1 \mathbf{R} \psi_2$ и, следовательно, $\psi_2 \in B_{n+i}$.

Тогда по индуктивному предположению $tr|_{n+i} \models \psi_2$ для любого i, $i \geq 0$. Следовательно, $tr|_n \models \psi_1 \mathbf{R} \psi_2$.



ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Индуктивный переход.

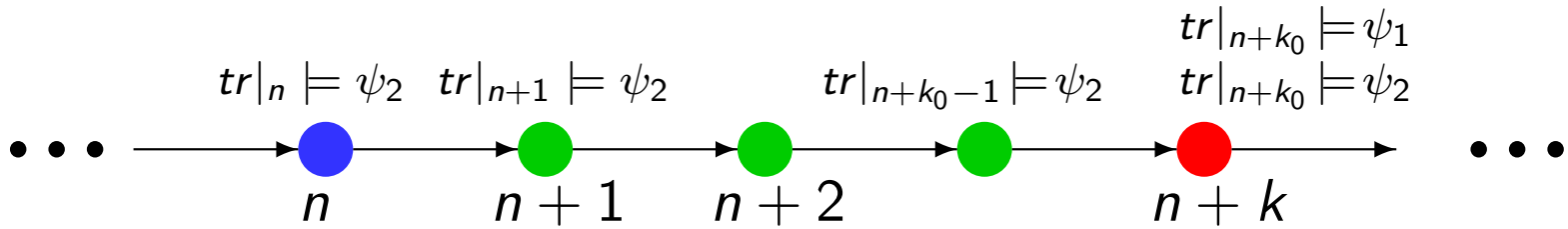
Вариант 2. $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \notin B_{n+k}$ для некоторого k, $k \geq 0$.

Тогда существует такое k_0, что $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \notin B_{n+k_0}$ но $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \in B_{n+i}$ для любого i, $0 \leq i < k_0$.

Тогда по определению графа $\Gamma_{\varphi_1, M}$ в каждом множестве B_{n+i}, $0 \leq i \leq k_0$, содержится формула $\psi_1 \mathbf{R} \psi_2$.

Тогда по определению согласованных множеств $\psi_2 \in B_{n+i}$ для любого i, $0 \leq i \leq k_0$, и, кроме того, $\psi_1 \in B_{n+k_0}$.

Тогда по индуктивному предположению $tr|_{n+i} \models \psi_2$ для любого $0 \leq i \leq k_0$ и $tr|_{n+k_0} \models \psi_1$. Значит, $tr|_n \models \psi_1 \mathbf{R} \psi_2$.



Итак, в обоих случаях $\psi_1 \mathbf{R} \psi_2 \in B_n \Longrightarrow tr|_n \models \psi_1 \mathbf{R} \psi_2$.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Индуктивный переход.

3.2. Покажем $\psi_1 \mathbf{R} \psi_2 \notin B_n \Longrightarrow tr|_n \not\models \psi_1 \mathbf{R} \psi_2$.

Пусть $\psi_1 \mathbf{R} \psi_2 \notin B_n$. Т. к. $\psi_1 \mathbf{R} \psi_2 \in URSub_{\varphi_1}$ этой формуле соответствует некоторый цвет j.
Поскольку рассматриваемый маршрут

$$(s_0, B_0), (s_1, B_1), \ldots, (s_n, B_n), (s_{n+1}, B_{n+1}), \ldots$$

является радужным, то вершины, окрашенные в цвет j, встречаются в этом маршруте бесконечно часто.
Значит, существует такое k, $k \geq 0$, что вершина (s_{n+k}, B_{n+k}) — первая, окрашенная в цвет j вершина, следующая в этом радужном маршруте вслед за вершиной (s_n, B_n).

Имеются две причины, по которым вершина (s_{n+k}, B_{n+k}) оказалась окрашенной в цвет j:

- $\psi_2 \notin B_{n+k}$,
- $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \in B_{n+k}$,

Рассмотрим каждый из этих случаев по отдельности.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Индуктивный переход.

Вариант 1. $\psi_2 \notin B_{n+k}$.

Т. к. все вершины (s_{n+i}, B_{n+i}), $0 \leq i < k$ не окрашены в цвет j, для каждого из множеств B_{n+i}, $0 \leq i < k$, верны соотношения

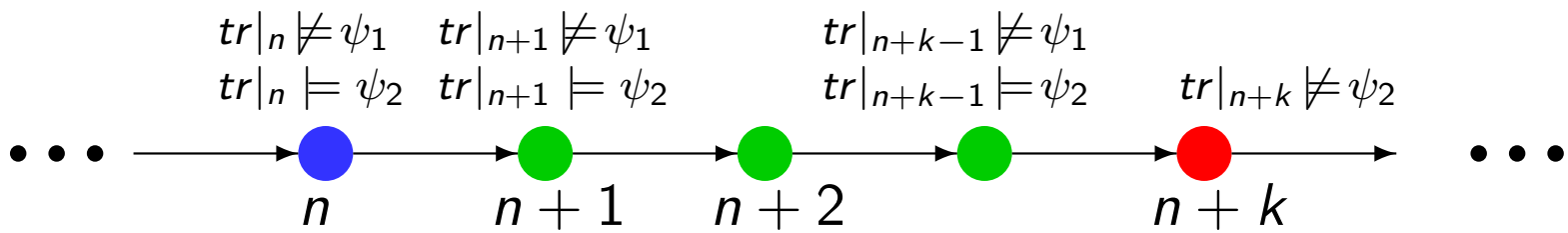
$$\psi_2 \in B_{n+i} \qquad \text{и} \qquad \mathbf{X}(\psi_1 \mathbf{R} \psi_2) \notin B_{n+i}.$$

Тогда по определению графа $\Gamma_{\varphi_1, M}$ для каждого множества B_{n+i}, $0 \leq i < k$, верно соотношение $\psi_1 \mathbf{R} \psi_2 \notin B_{n+i}$. А отсюда следует, что $\psi_1 \notin B_{n+i}$ для любого i, $0 \leq i < k$.

Тогда по индуктивному предположению

$tr|_{n+i} \models \psi_2$ и $tr|_{n+i} \not\models \psi_1$ для любого i, $0 \leq i < k$,

$tr|_{n+k} \not\models \psi_2$.



А это означает, что $tr|_n \not\models \psi_1 \mathbf{R} \psi_2$.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Индуктивный переход.

Вариант 2. $\mathbf{X}(\psi_1 \mathbf{R} \psi_2) \in B_{n+k}$.

Т. к. все вершины (s_{n+i}, B_{n+i}), $0 \leq i < k$ не окрашены в цвет j, для каждого из множеств B_{n+i}, $0 \leq i < k$, верны соотношения

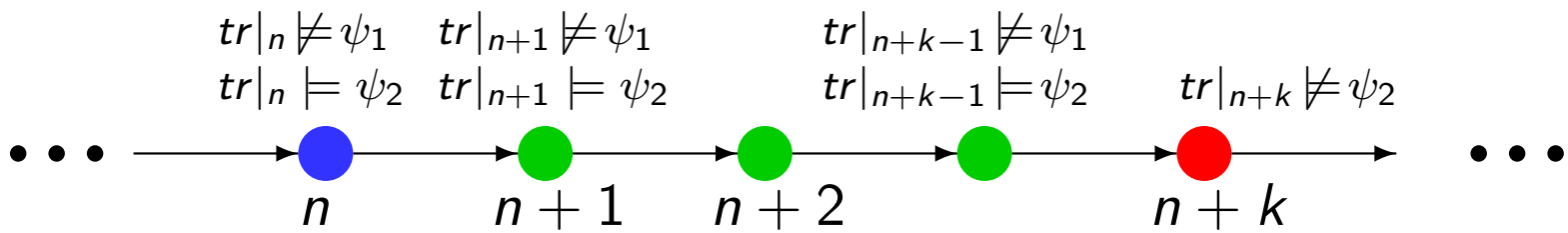
$$\psi_2 \in B_{n+i} \qquad \text{и} \qquad \mathbf{X}(\psi_1 \mathbf{R} \psi_2) \notin B_{n+i}.$$

Тогда по определению графа $\Gamma_{\varphi_1, M}$ для каждого множества B_{n+i}, $0 \leq i \leq k$, верно соотношение $\psi_1 \mathbf{R} \psi_2 \notin B_{n+i}$. А отсюда следует, что $\psi_1 \notin B_{n+i}$ для любого i, $0 \leq i < k$ и $\psi_2 \notin B_{n+k}$.

Тогда по индуктивному предположению

$tr|_{n+i} \models \psi_2$ и $tr|_{n+i} \not\models \psi_1$ для любого i, $0 \leq i < k$,

$tr|_{n+k} \not\models \psi_2$.



И во втором случае $tr|_n \not\models \psi_1 \mathbf{R} \psi_2$.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Индуктивный переход.

Таким образом, если $\psi_1 \mathbf{R} \psi_2 \notin B_n$, то $tr|_n \not\models \psi_1 \mathbf{R} \psi_2$.

В итоге, для любой формулы вида $\psi_1 \mathbf{R} \psi_2$ и для любой вершины (s_n, B_n) нашего **радужного** маршрута верно соотношение

$$\psi_1 \mathbf{R} \psi_2 \in B_n \iff tr|_n \models \psi_1 \mathbf{R} \psi_2 \ .$$

4. Темпоральный оператор **U**.

Для доказательства соотношения

$$\psi_1 \mathbf{R} \psi_2 \in B_n \iff tr|_n \models \psi_1 \mathbf{R} \psi_2$$

применяются рассуждения, аналогичные тем, которые были использованы для исследования оператора **R**.

Самостоятельно.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Завершив обоснование индуктивного перехода, мы тем самым завершили доказательство первой части теоремы:

$$M \not\models \varphi_1$$

$$\Uparrow$$

в графе $\Gamma_{\varphi_1,M}$ существует хотя бы один радужный маршрут, исходящий из вершины $v_0 = (s_0, B_0)$, в которой $s_0 \in S_0$ и $\varphi_1 \notin B_0$.

Покажем, что в том случае, когда имеет место $M \not\models \varphi_1$, в графе $\Gamma_{\varphi_1,M}$ из некоторой вершины $v_0 = (s_0, B_0)$, в которой $s_0 \in S_0$ и $\varphi_1 \notin B_0$, исходит хотя бы один радужный маршрут.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Пусть $M \not\models \varphi_1$. Тогда в LTS M существует такая начальная трасса tr, для которой $tr \not\models \varphi_1$. Рассмотрим эту трассу tr.

Для каждого i, $i \geq 0$, положим

$$B_i = \{\psi \ : \ \psi \in \mathit{FLSub}_{\varphi_1}, tr|_i \models \psi \ .\}$$

Согласно утверждению 4, все построенные множества B_i являются согласованными.

Покажем, что последовательность пар

$$(tr[0], B_0), (tr[1], B_1), (tr[2], B_2), \ldots, (tr[n], B_n), (tr[n{+}1], B_{n+1}), \ldots$$

образует искомый радужный маршрут в графе Γ_{φ_1}.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Действительно,

1. Для любого n, $n \geq 0$, верно $tr[n] \longrightarrow tr[n+1]$, поскольку tr — маршрут в LTS M.

2. Для любого n, $n \geq 0$ и для любой формулы $\mathbf{X}\psi \in XSub_{\varphi_1}$, верно

$$\mathbf{X}\psi \in B_n \iff \psi \in B_{n+1}$$

поскольку

$$\mathbf{X}\psi \in B_n \iff tr|_n \models \mathbf{X}\psi \iff tr|_{n+1} \models \psi \iff \psi \in B_{n+1} \ .$$

3. $tr[0] \in S_0$ (т. к. tr — начальная трасса в M) и $\varphi_1 \notin B_0$ (т. к. $tr|_0 \not\models \varphi_1$).

Значит, последовательность

$$(tr[0], B_0), (tr[1], B_1), (tr[2], B_2), \ldots, (tr[n], B_n), (tr[n+1], B_{n+1}), \ldots$$

является маршрутом в графе $\Gamma_{\varphi_1, M}$, исходящим из нужной вершины.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

4. Осталось показать, что маршрут

$$(tr[0], B_0), (tr[1], B_1), (tr[2], B_2), \ldots, (tr[n], B_n), (tr[n{+}1], B_{n+1}), \ldots$$

является радужным.

Рассмотрим произвольное число n, $n \geq 0$ и произвольную формулу $\psi_i \in URSub_{\varphi_1}$. Покажем, что существует такое $k, k \geq 0$, что вершина $(tr[n+k], B_{n+k})$ окрашена в цвет i.

ТАБЛИЧНЫЙ МЕТОД ВЕРИФИКАЦИИ

Ограничимся рассмотрением Until-формулы $\psi_i = \chi' \mathbf{U} \chi_2$. (Для формул вида $\psi_i = \chi' \mathbf{R} \chi_2$ доказательство проведите самостоятельно.)

1. Если $tr|_n \not\models \mathbf{X}(\chi_1 \mathbf{U} \chi_2)$, то $\mathbf{X}(\chi_1 \mathbf{U} \chi_2) \notin B_n$, и, следовательно, вершина $(tr[n], B_n)$ окрашена в цвет j.
2. А если $tr|_n \models \mathbf{X}(\chi_1 \mathbf{U} \chi_2)$, то $tr|_{n+1} \models \chi_1 \mathbf{U} \chi_2$. Тогда существует такое k, $k \geq 1$, что $tr|_{n+k} \models \chi_2$. Поэтому $\chi_2 \in B_{n+k}$, и вершина $(tr[n+k], B_{n+k})$ окрашена в цвет j.

Таким образом, вершины цвета j встречаются в нашем маршруте бесконечно часто. Поскольку ψ_i была произвольной (Until-Release)-формулой, это означает, что наш маршрут в графе $\Gamma_{\varphi_1, M}$ является радужным. □

АЛГОРИТМ ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Но как проверить, что из заданной вершины в графе $\Gamma_{\varphi_1,M}$ не исходит ни одного радужного маршрута?

Ориентированный граф Г называется сильно связным , если для любой пары вершин v и u в графе Г существует маршрут из v в u и маршрут из u в v.

Всякий максимальный сильно связный подграф графа Г называется компонентой сильной связности .

Компоненту сильной связности графа (системы Хинтикки) $\Gamma_{\varphi_1,M}$ будем называть радужной, если в ней содержатся вершины **всех** цветов.

АЛГОРИТМ ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Теорема.

Из вершины v в графе $\Gamma_{\varphi_1,M}$ исходит радужный маршрут тогда и только тогда, когда существует маршрут , ведущий из вершины v хотя бы в одну из вершин хотя бы одной радужной компоненты сильной связности.

Доказательство.

Самостоятельно. Здесь все очевидно.

АЛГОРИТМ ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Исходные данные: формула PLTL φ и LTS $M = \langle \mathcal{AP}, S, S_0, \longrightarrow, \rho \rangle$.

1. Построить равносильную позитивную форму φ_1.
2. Построить систему Хинтикки $\Gamma_{\varphi_1, M}$.
3. Выделить множество подформул $URSub_{\varphi_1}$ и раскрасить вершины графа $\Gamma_{\varphi_1, M}$.
4. Выделить радужные компоненты сильной связности в графе $\Gamma_{\varphi_1, M}$.
5. Выделить множество V' всех вершин графа $\Gamma_{\varphi_1, M}$, из которых достижимы радужные компоненты сильной связности.
6. Выделить множество V'' всех вершин (s_0, B_0), для которой выполняется $s_0 \in S_0$, $\varphi_1 \notin B_0$.
7. Вычислить $V = V' \cap V''$.

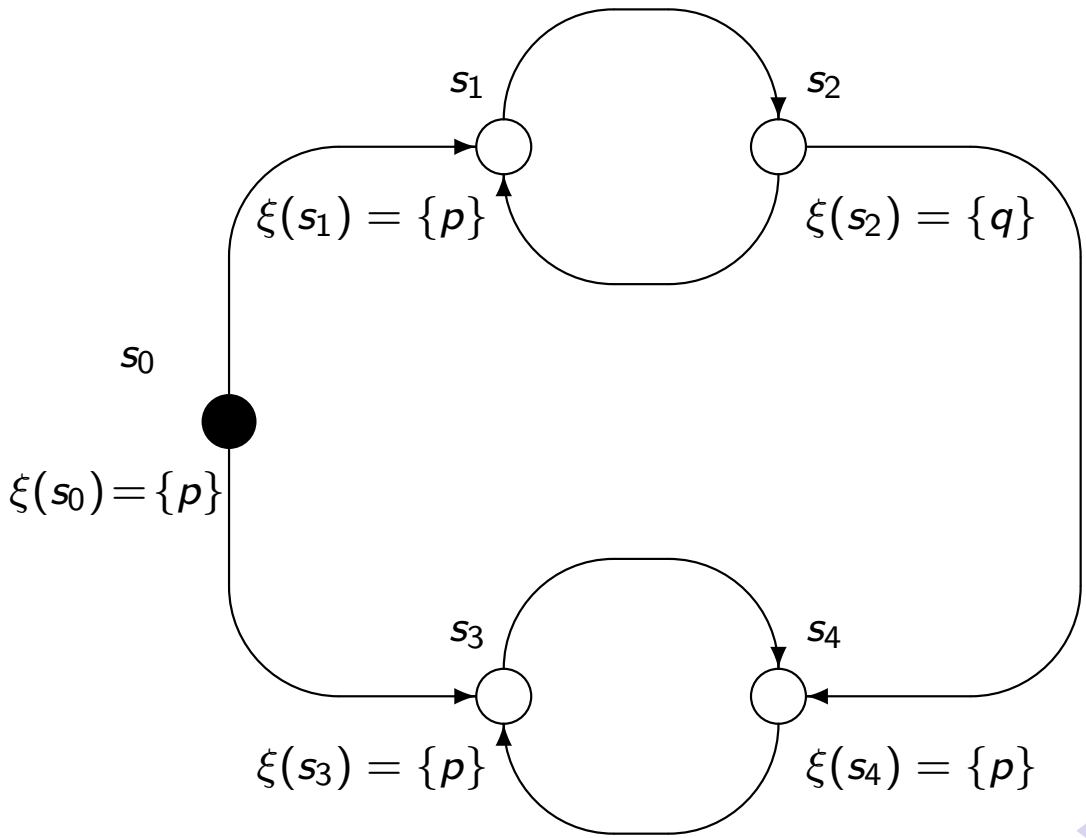
Результат: $M \models \varphi \iff V = \varnothing$.

АЛГОРИТМ ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Пример.

$$\varphi = p \mathbf{U} q$$

LTS M:



АЛГОРИТМ ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Пример.

$\varphi = \mathbf{F}(p\mathbf{U}q)$

1. Позитивная форма $\varphi_1 = p\mathbf{U}q$

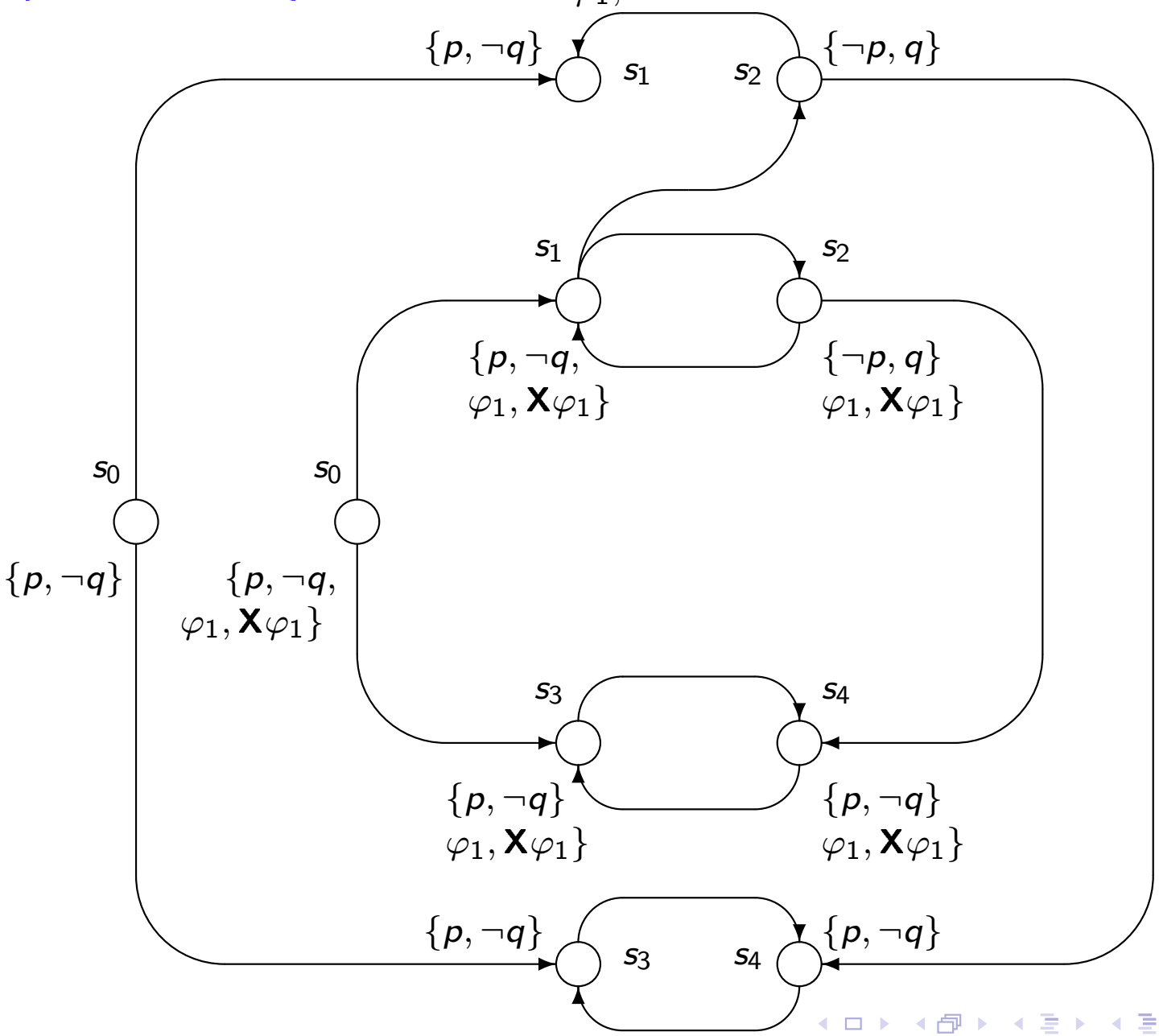
$FLSub_{\varphi_1} = \{p, \neg p, q, \neg q, p\mathbf{U}q, \mathbf{X}(p\mathbf{U}q)\}$;

$XSub_{\varphi_1} = \{\mathbf{X}(p\mathbf{U}q)\}$;

$URSub_{\varphi_1} = \{p\mathbf{U}q\}$.

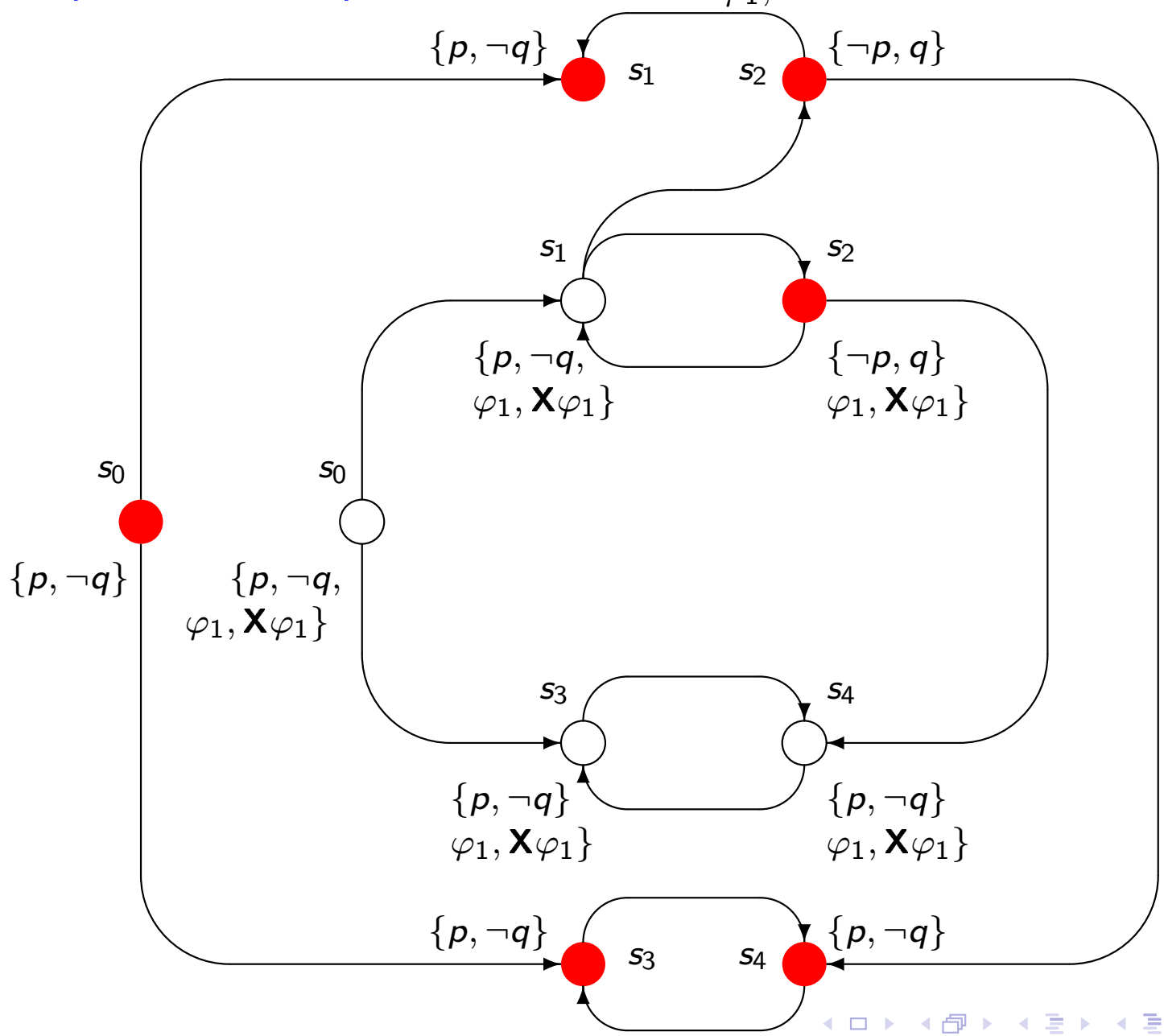
АЛГОРИТМ ВЕРИФИКАЦИИ

2. Строим систему Хинтикки $\Gamma_{\varphi_1, M}$



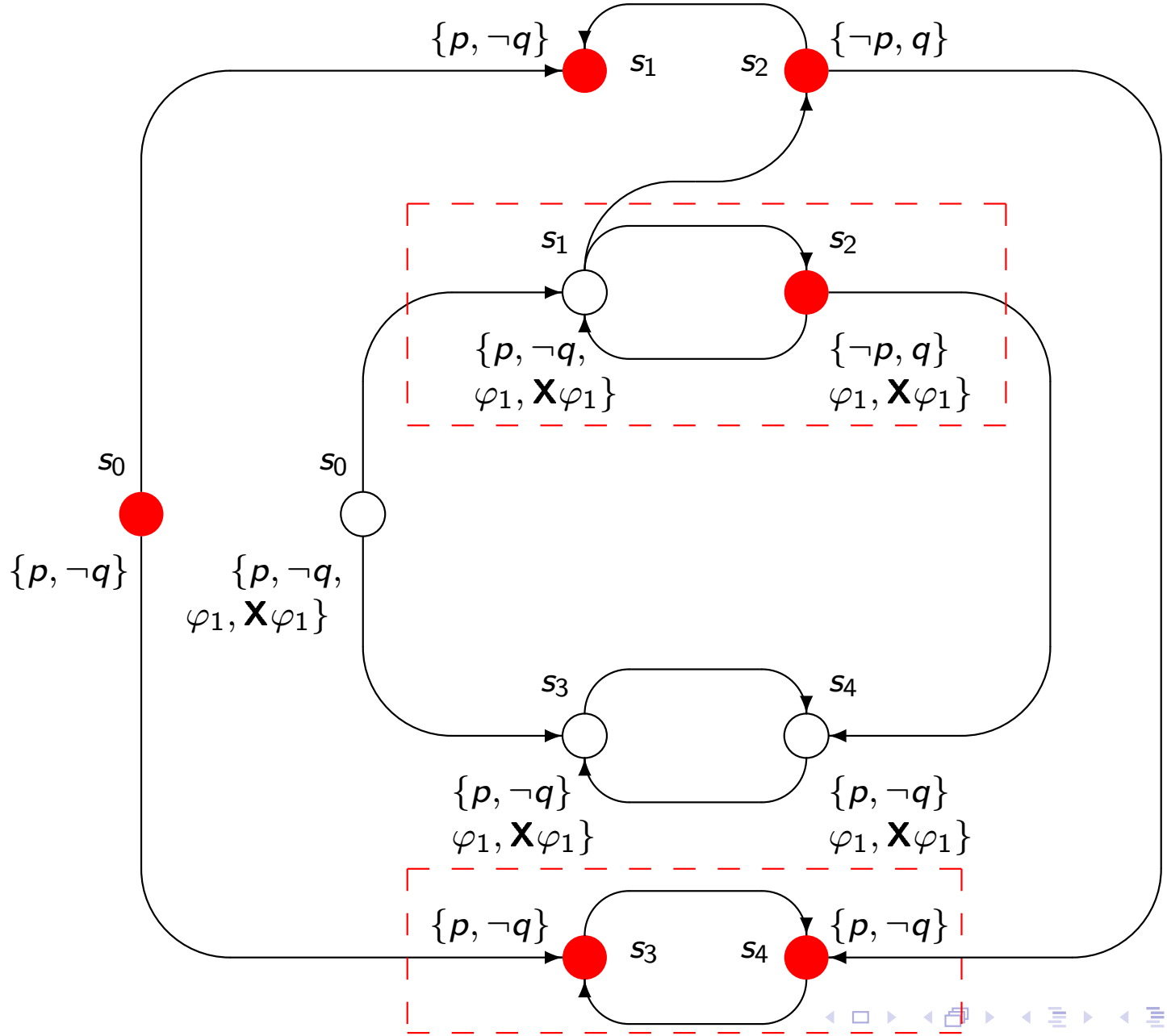
АЛГОРИТМ ВЕРИФИКАЦИИ

3. Раскрашиваем вершины системы $\Gamma_{\varphi_1, M}$



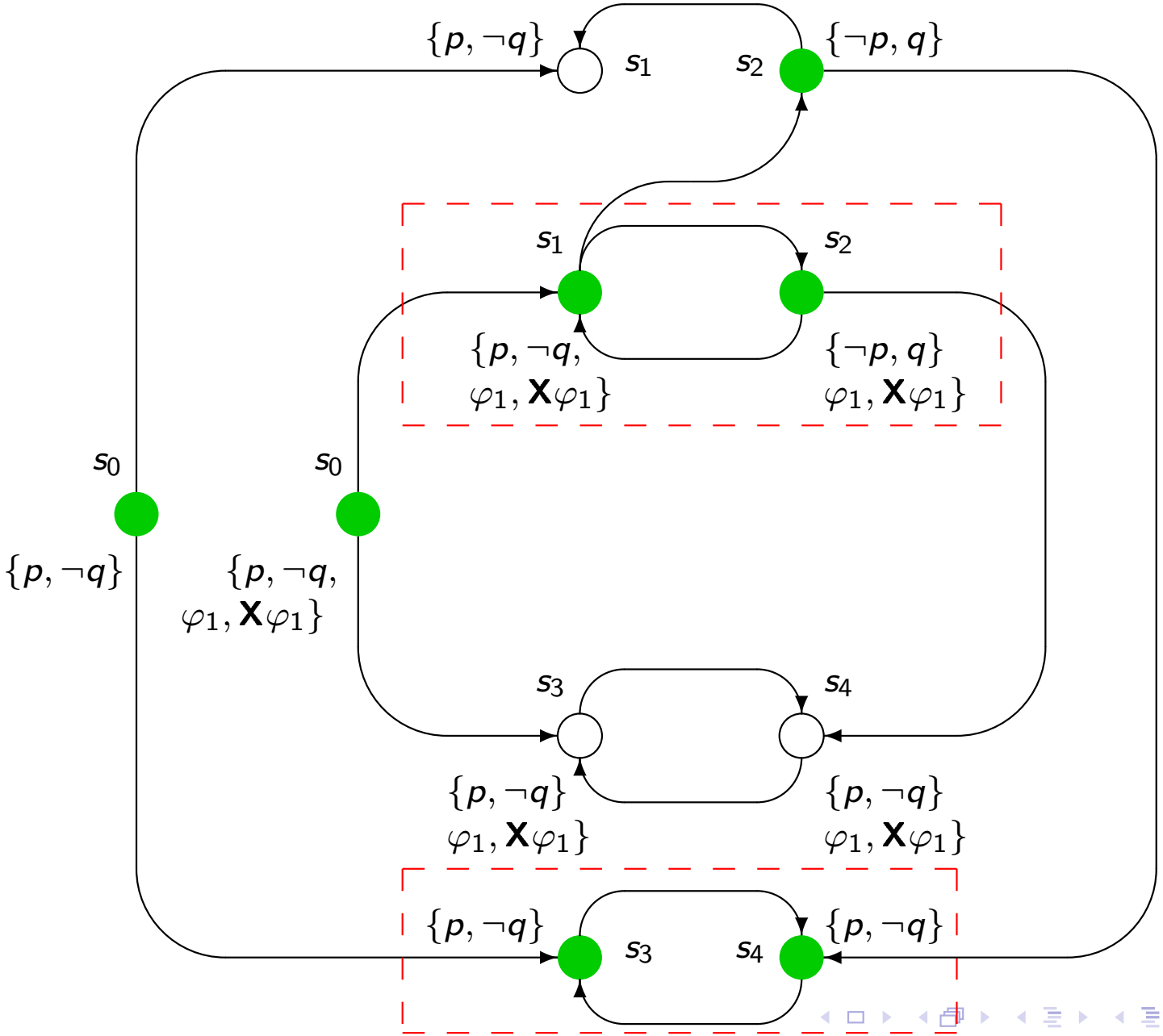
АЛГОРИТМ ВЕРИФИКАЦИИ

4. Выделяем радужные компоненты сильной связности



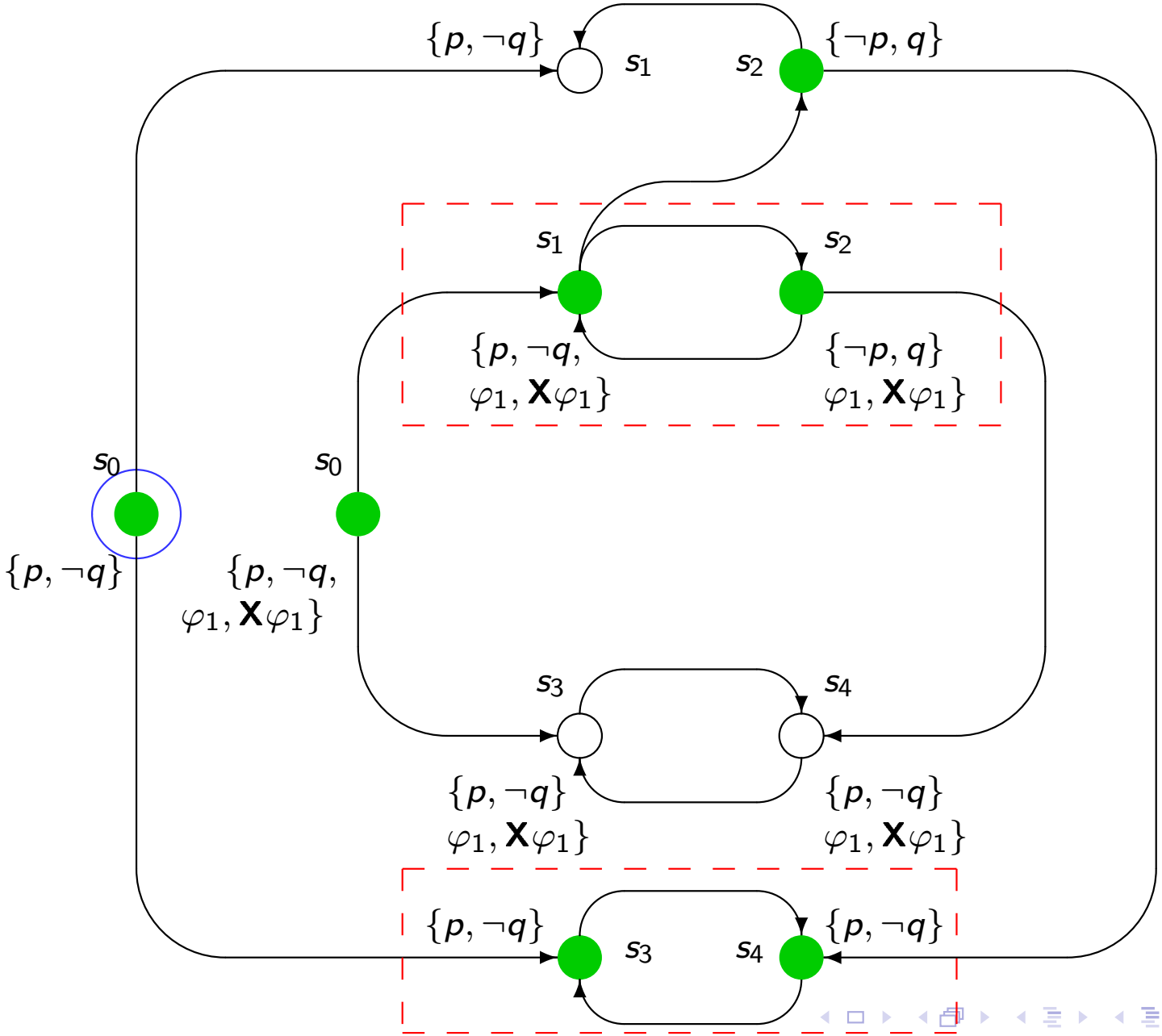
АЛГОРИТМ ВЕРИФИКАЦИИ

5. Выделяем вершины из которых достижимы радужные компоненты



АЛГОРИТМ ВЕРИФИКАЦИИ

6. Ищем вершину (s_0, B) на которой опровергается φ_1



АЛГОРИТМ ВЕРИФИКАЦИИ МОДЕЛЕЙ ПРОГРАММ

Описанный здесь подход к верификации распределенных программ реализован в программно-инструментальной системе верификации **SPIN** .

Модели параллельных взаимодействующих процессов описываются на языке **PROMELA** (Process Meta Language), снабжаются темпоральными спецификациями (PLTL формулами), а затем выполнимость этих формул проверяется системой верификации **SPIN** .

В системе **SPIN** применяется табличный алгоритм верификации моделей распределенных программ. Для повышения его эффективности используется ряд приемов:

- проверка модели «на лету»;
- редукции частичных порядков;
- символьное представление данных и др.

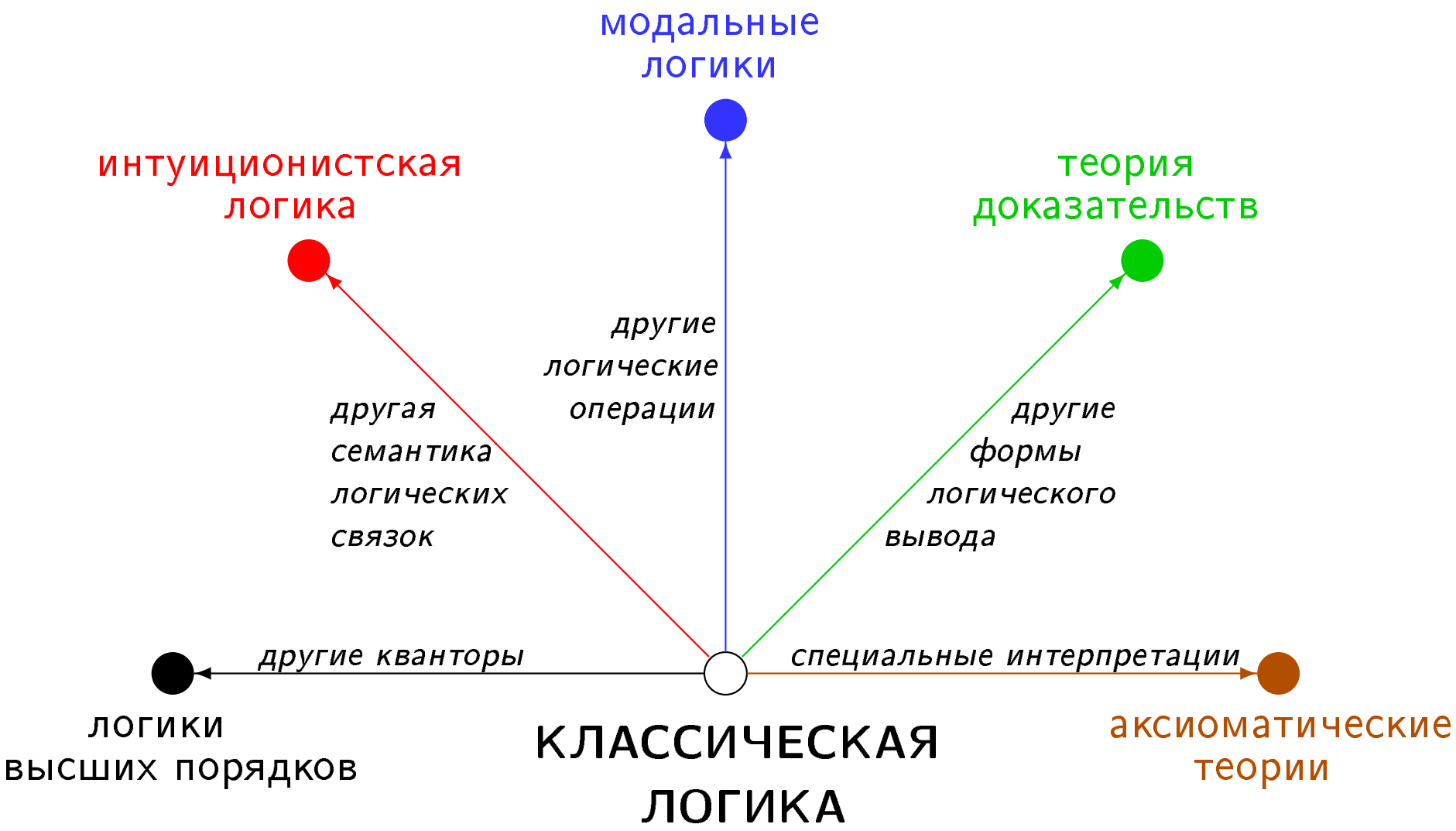
КОНЕЦ ЛЕКЦИИ 22

Основы математической логики и логического программирования

ЛЕКТОР: В.А. Захаров

Лекция 23.

Как устроена математика.
Исчисление предикатов
первого порядка.
Аксиоматические теории.
Элементарная геометрия.
Теория множеств
Цермело–Френкеля.
Арифметика Пеано.
Теорема Геделя о неполноте.


модальные
логики
интуиционистская
логика
теория
доказательств
другие
логические
операции
другая
семантика
логических
связок
другие
формы
логического
вывода
другие кванторы
специальные интерпретации
логики
высших порядков
КЛАССИЧЕСКАЯ
ЛОГИКА
аксиоматические
теории

Как устроена математика

Математика — это специфическая наука.

Она не относится к числу естественных наук (физика, ботаника, геология, и пр.), т. к. она не имеет дела ни с природными явлениями, ни с эмпирическими знаниями.

Она не относится к числу гуманитарных наук (философия, история, политология и пр. болтология), т. к она не занимается ни людской деятельностью, ни людскими воззрениями.

Она занимается созданием, развитием и изучением математических теорий — умозрительных конструкций, которые строятся по строгим объективным законам формальной логики .

Как устроена математика

Станислав Лем сравнивал математику с безумным портным, который шьет одежду для неведомых существ.

Портного не беспокоит, кому придется впору его одежда.

Он лишь хочет, чтобы платье было сшито прочно.

Как устроена математика

С чего начинается рассказ о каждом разделе математики?

- Вначале уславливаются о системе обозначений, определяют язык, на котором будут записывать математические утверждения (определяется синтаксис математического языка).
- Затем приходят к соглашению об основополагающих свойствах, законах, которым должны удовлетворять интересующие нас операции и отношения над воображаемыми объектами (формулируются аксиомы математической теории).
- Далее договариваются о том, какие средства обоснования истинности математических утверждений считаются допустимыми (определяется аппарат логического вывода).
- И после этого приступают к получению логически обоснованных утверждений сформулированной математической теории (вывод теорем).

Вот так строятся формальные аксиоматические теории.

Классическое исчисление предикатов

Как можно аксиоматизировать теорию общезначимых утверждений (формул)? Например, так:

АКСИОМЫ.

1. Ax1. $\varphi_1 \to (\varphi_2 \to \varphi_1)$,
2. Ax2. $(\varphi_1 \to (\varphi_2 \to \varphi_3)) \to ((\varphi_1 \to \varphi_2) \to (\varphi_1 \to \varphi_3))$,
3. Ax3. $(\varphi_1 \,\&\, \varphi_2) \to \varphi_1$,
4. Ax4. $(\varphi_1 \,\&\, \varphi_2) \to \varphi_2$,
5. Ax5. $\varphi_1 \to (\varphi_2 \to (\varphi_1 \,\&\, \varphi_2))$,
6. Ax6. $\varphi_1 \to (\varphi_1 \vee \varphi_2)$,
7. Ax7. $\varphi_2 \to (\varphi_1 \vee \varphi_2)$,
8. Ax8. $(\varphi_1 \to \varphi_0) \to ((\varphi_2 \to \varphi_0) \to ((\varphi_1 \vee \varphi_2) \to \varphi_0))$,
9. Ax9. $\varphi_1 \to (\neg\varphi_1 \to \varphi_0)$,
10. Ax10. $\varphi_1 \vee \neg\varphi_1$,

Классическое исчисление предикатов

АКСИОМЫ.

1. Ax11. $\forall X\ \varphi(X) \to \varphi(t)$,
2. Ax12. $\varphi(t) \to \exists X\ \varphi(X)$,
3. Ax13. $\forall X\ (\varphi_1 \to \varphi_2(X)) \to (\varphi_1 \to \forall X\ \varphi_2(X))$,
4. Ax14. $\forall X\ (\varphi_1(X) \to \varphi_2) \to (\exists X\ \varphi_1(X) \to \varphi_2)$.

ПРАВИЛА ВЫВОДА.

1. Правило отделения (modus ponens) $\dfrac{\varphi,\ \varphi \to \psi}{\psi}$,
2. Правило обобщения $\dfrac{\varphi}{\forall X\ \varphi}$

Классическое исчисление предикатов

ЛОГИЧЕСКИЙ ВЫВОД.

Пусть задано некоторое множество формул (гипотез) Γ.
Тогда логическим выводом из множества гипотез Γ называется конечная последовательность формул

$$\varphi_1, \varphi_2, \ldots, \varphi_n,$$

в которой каждая формула φ_i удовлетворяет одному из следующих условий:

1. либо φ_i является аксиомой,
2. либо φ_i является гипотезой, т. е. $\varphi_i \in \Gamma$,
3. либо φ_i получается из предшествующих формул этой последовательности по правилу отделения или по правилу обобщения.

В этом случае формула φ_n называется выводимой из множества Γ, и этот факт обозначается $\Gamma \vdash \varphi_n$

Формула φ называется теоремой, если $\varnothing \vdash \varphi$, и этот факт обозначается $\vdash \varphi$.

Классическое исчисление предикатов

Исчисление предикатов с равенством.

Введем специальный двухместный предикатный символ $=$ и добавим к аксиомам КИП следующие аксиомы равенства:

1. Ax15. $\forall X\ (X = X)$,
2. Ax16 $\forall X, Y\ (X = Y\ \rightarrow\ (\varphi(X, X)\ \rightarrow\ \varphi(X, Y)))$.

Полученную систему аксиом называют классическим исчислением предикатов с равенством $КИП_{=}$.

Алгебраическая система I называется нормальной интерпретацией, если для любой пары различных предметов d_1, d_2 из области интерпретации D_I верно соотношение

$$I\ \models\ d_1 = d_2\ .$$

Аксиоматические теории первого порядка

Элементарная аксиоматическая теория образуется из исчисления предикатов с равенством за счет

- ограничения сигнатуры языка логики предикатов фиксированным конечным набором констант, функциональных и предикатных символов, обозначающих базовые объекты, операции и отношения теории,
- добавления к множеству аксиом исчисления предикатов специальных (нелогических) аксиом, описывающих базовые принципы теории.

Таким образом образуются элементарная теория равенства, элементарная теория групп, элементарная теория полей, элементарная геометрия, элементарная арифметика, элементарная теория множеств, и др.

Формулы φ, логически выводимые из аксиом элементарной аксиоматической теории T, называются теоремами теории T и обозначаются записью $T \vdash \varphi$.

Аксиоматические теории первого порядка

Элементарная аксиоматическая теория T называется

- непротиворечивой, если не все формулы являются теоремами теории T, т. е. существует такая формула φ, для которой $T \nvdash \varphi$;
- полной, если для всякой замкнутой формулы либо она сама, либо ее отрицание является теоремой теории T, т. е. для любой формулы φ либо $T \vdash \varphi$, либо $T \vdash \neg\varphi$;
- категоричной, если любые две **нормальные** модели теории T изоморфны, т. е. для любой пары нормальных интерпретаций I_1, I_2 верно
$$I_1 \models T \text{ и } I_2 \models T \implies I_1 \cong I_2;$$
- разрешимой, если существует алгоритм, проверяющий, является ли произвольная формула теоремой теории T.

Утверждение.

Всякая полная конечно аксиоматизируемая теория разрешима.

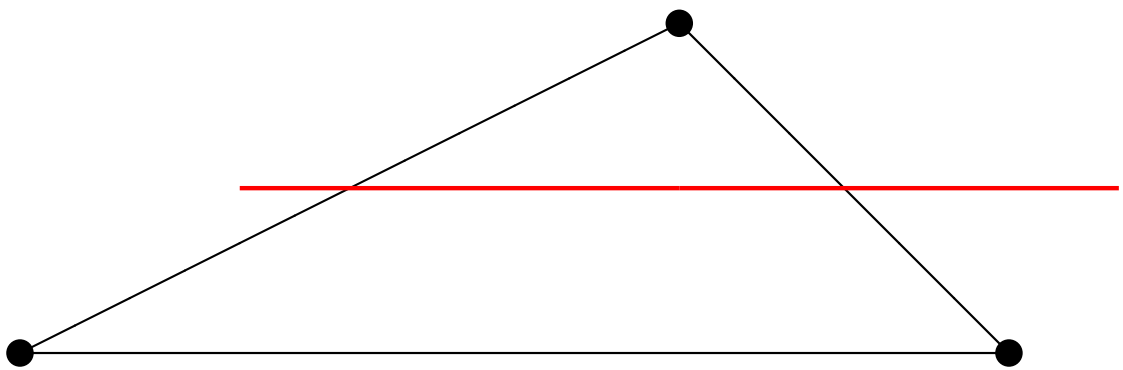
Аксиоматическое устройство геометрии

Впервые попытку аксиоматизировать геометрию предпринял Евклид (3 в. до н. э.). Геометрическая теория Евклида опиралась на 5 аксиом.

К сожалению, система геометрических аксиом из «Начал» Евклида неполна.

Вот пример истинного утверждения, которое нельзя вывести из аксиом и постулатов Евклида.

Если прямая пересекает одну из сторон треугольника в точке, отличной от вершины треугольника, то эта прямая также пересекает еще одну сторону треугольника.



Аксиоматическое устройство геометрии

Систематическое и основательное построение геометрической системы аксиом было осуществлено Д. Гильбертом (40 аксиом) в 1899 г. Более более краткую аксиоматику удалось построить А. Тарскому и его ученика (12 аксиом).

Аксиомы Тарского

Будем рассматривать геометрический мир, все объекты которого — точки .

На множестве точек есть всего лишь два базовых предиката:

$\mathbf{B(x, y, z)}$	точка y лежит между точками x и z на одной прямой
$\mathbf{D(x, y, z, u)}$	точка x отстоит от точки y на такое же расстояние, что и точка z от точки u

Аксиоматическое устройство геометрии

Аксиомы T1–T5

1). $\forall x, y, z\ (B(x, y, z) \rightarrow B(z, y, x))$
(аксиома симметричности предиката B)

2). $\forall x, y, z, u\ (B(x, y, u)\&B(y, z, u) \rightarrow B(x, y, z))$
(аксиома транзитивности предиката B)

3). $\forall x, y\ D(x, y, y, x)$
(аксиома симметричности равенства длин отрезков)

4). $\forall x, y, z\ (D(x, y, z, z) \rightarrow x = y)$
(аксиома нулевого отрезка)

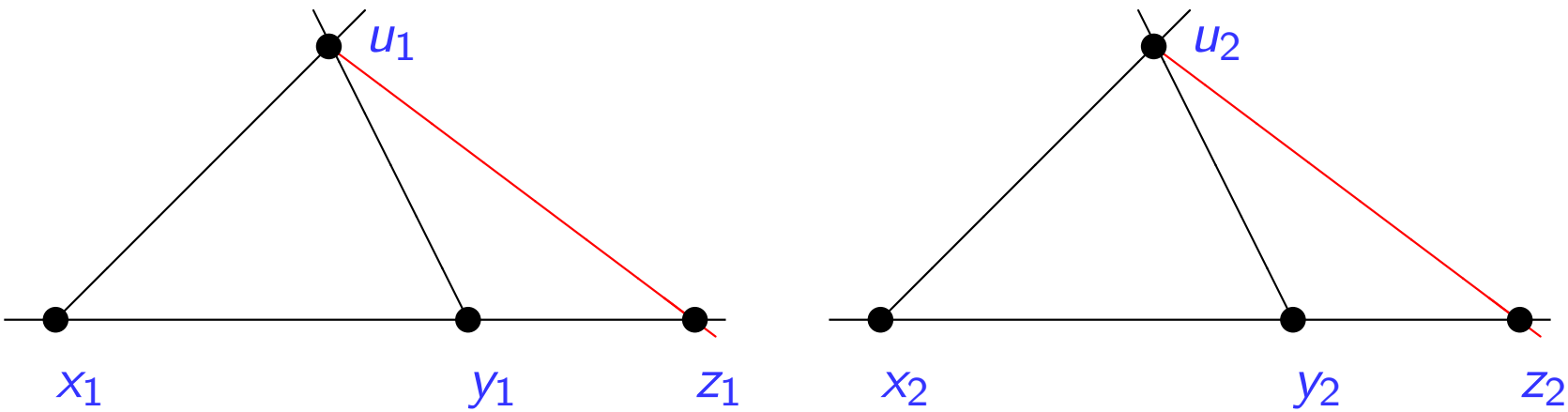
5). $\forall x_1, y_1, x_2, y_2, x_3, y_3$
$(D(x_1, y_1, x_2, y_2)\&D(x_2, y_2, x_3, y_3) \rightarrow D(x_1, y_1, x_3, y_3))$
(аксиома транзитивности равенства длин отрезков)

Аксиоматическое устройство геометрии

Аксиома T6

6). $\forall x_1, y_1, z_1, u_1, x_2, y_2, z_2, u_2$
$(x_1 \neq y_1 \& y_1 \neq z_1 \&$
$B(x_1, y_1, z_1) \& B(x_2, y_2, z_2) \&$
$D(x_1, y_1, x_2, y_2) \& D(y_1, z_1, y_2, z_2) \&$
$D(y_1, u_1, y_2, u_2) \& D(x_1, u_1, x_2, u_2) \rightarrow$
$\rightarrow D(z_1, u_1, z_2, u_2))$

(аксиома пяти отрезков)



Аксиоматическое устройство геометрии

Аксиомы T7–T10

7). $\forall x, y, z, u\ \exists v\ (B(x, y, v)\ \&\ D(y, v, z, u))$
(аксиома откладывания отрезка)

8). $\forall x, y\ \exists z\ (B(x, z, y) \& D(x, z, z, y))$
(аксиома деления отрезка пополам)

9). $\exists x, y, z\ (\neg B(x, y, z)\ \&\ \neg B(x, z, y)\ \&\ \neg B(z, x, y))$
(аксиома существования неколлинеарных точек)

10). $\forall x, y, z\ (\neg B(x, y, z)\ \&\ \neg B(x, z, y)\ \&\ \neg B(z, x, y)\ \rightarrow$
$\rightarrow\ \exists v\ (D(v, x, v, y) \& D(v, x, v, z)))$
(аксиома центра описанной окружности)

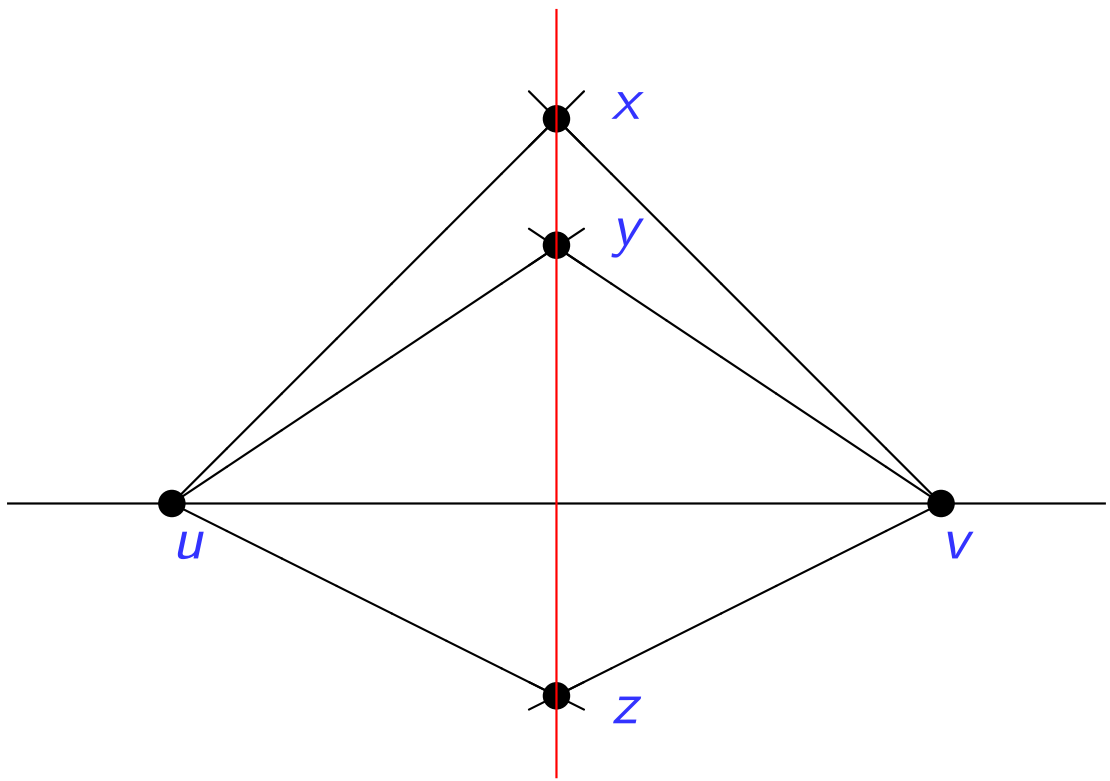
Аксиоматическое устройство геометрии

Аксиома T11

11). $\forall x, y, z, u, v$

$$(D(x,u,x,v)\&D(y,u,y,v)\&D(z,u,z,v) \rightarrow$$
$$\rightarrow (B(x,y,z) \vee B(y,z,x) \vee B(z,y,x)))$$

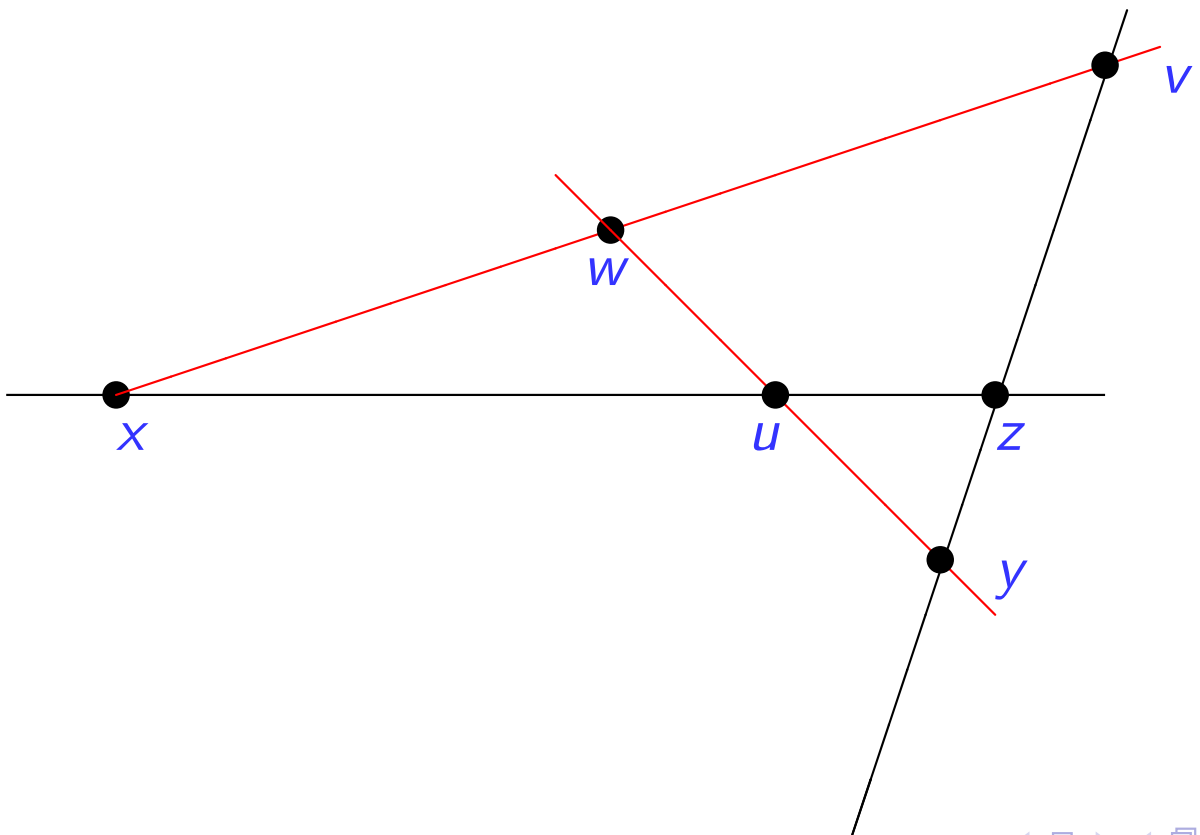
(аксиома перпендикуляра к середине отрезка)



Аксиоматическое устройство геометрии

Аксиома T12

12). $\forall x, y, z, u, v$
$(B(x, u, z) \& B(y, z, v) \rightarrow$
$\rightarrow \exists w\ (B(y, u, w) \& B(x, w, v)))$
(аксиома Паша)



Аксиоматическое устройство геометрии

Аксиома T13

13). $\exists x\ \forall y, z\ (\varphi(y)\&\psi(z)\ \rightarrow\ B(x, y, z))\ \rightarrow$
$\rightarrow\ \exists x'\ \forall y, z\ (\varphi(y)\&\psi(z)\ \rightarrow\ B(y, x', z))$
(схема аксиом непрерывности)

Аксиоматическое устройство геометрии

Основные свойства формальной геометрии Тарского

Теорема

Аксиоматическая теория T1–T13 (формальная геометрия Тарского)

- непротиворечива,
- полна,
- категорична,
- алгоритмически разрешима.

К сожалению для школьников, разрешающая процедура, способная доказывать любую геометрическую теорему, имеет невероятно большую вычислительную сложность.

Теория множеств

МНОЖЕСТВО — это основополагающее понятие современной математики. Понятие множества предложил во второй половине 19 в. немецкий математик Георг Кантор.

А что же такое множество?

Поскольку это основополагающее понятие, строгого определения дать нельзя. Это коллекция (семейство, совокупность, собрание) различных предметов (объектов, элементов).

Может ли математика спокойно развиваться, опираясь на столь зыбкое основание?

Теория множеств

Парадокс Рассела

Элементами множеств могут быть множества. Рассмотрим коллекцию всех множеств, каждое из которых не является своим собственным элементом: $A = \{x : x \notin x\}$.
У нас нет достаточных оснований не признавать эту совокупность множеств A множеством.

Но тогда мы должны уметь давать ответ на вопрос: содержит ли множество A в качестве элемента само множество A (т. е. верно ли что $A \in A$?)
Ответ обескураживающий:

- если $A \in A$, то по определению A верно $A \notin A$,
- а если $A \notin A$, то по определению A верно $A \in A$.

Теория множеств

Значит, в наивной теории множеств существуют математические утверждения, которые нельзя признать ни истинными, ни ложными. На основе такой расплывчатой теории хорошей математики не построить.

Может быть стоило бы исключить эту странную коллекцию A из числа множеств?

Можно. Но тогда придется создать «кодекс теории множеств», в котором должно быть указано, какие именно конструкции признаются множествами, и какими свойствами они должны обладать.

Попытку создания такого «кодекса теории множеств» — аксиоматической теории множеств — предпринял Эрнест Цермело в 1908 г.. Аксиоматику Цермело пополнили Абрахам Френкель, Торальф Сколем, Джон фон Нейман.

Теория множеств Цермело–Френкеля

Понятие множества и свойства множеств можно описать с использованием единственного предикатного символа $\in$, обозначающего отношение принадлежности одного множества в качестве элемента другого множества.

Представим себе математический мир, состоящий только из множеств. Этот мир может быть описан следующими аксиомами.

Теория множеств Цермело–Френкеля

1) $\forall x, y, u, v\ (x = y\ \&\ u = v)\ \rightarrow\ (x \in u\ \equiv\ y \in v)$
(Аксиома равенства множеств)
2) $\forall x, y\ (\forall z\ (z \in x\ \equiv\ z \in y)\ \equiv\ x = y)$
(Аксиома объемности)
3) $\forall x \forall u_1, \ldots, u_n\ \exists y\ \forall z\ (z \in y\ \equiv\ (z \in x\ \&\ \varphi(z, u_1, \ldots, u_n)))$
(Схема аксиом выделения)

здесь $\varphi(x, u_1, \ldots, u_n)$ — произвольная формула логики предикатов сигнатуры $\sigma = \langle \in \rangle$.

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

1. Существует единственное пересечение двух множество

Существование пересечения двух множеств следует из аксиомы выделения

$$\forall x_1, x_2 \; \exists y \; \forall z \; (z \in y \; \equiv \; (z \in x_1 \; \& \; z \in x_2)),$$

а единственность пересечения следует из аксиомы объемности

$$\forall x, y \; (\forall z \; (z \in x \; \equiv \; z \in y) \; \equiv \; x = y).$$

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

2. Существует единственное пустое множество

Существование пустого множества следует из аксимы выделения

$$\forall x\ \exists y \forall z (z \in y \ \equiv \ (z \in x \ \& \ z \neq z)),$$

а единственность пустого множества следует из аксиомы объемности

$$\forall x, y\ (\forall z\ (z \in x \ \equiv \ z \in y) \ \equiv \ x = y).$$

Но здесь есть один нюанс. А откуда берется множество X на основании которого определяется пустое множество?

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

2. Существует единственное пустое множество

Существование пустого множества следует из аксимы выделения

$$\forall X\ \exists y \forall z (z \in y\ \equiv\ (z \in X\ \&\ z \neq z)),$$

а единственность пустого множества следует из аксиомы объемности

$$\forall x, y\ (\forall z\ (z \in x\ \equiv\ z \in y)\ \equiv\ x = y).$$

Но здесь есть один нюанс. А откуда берется множество X на основании которого определяется пустое множество?

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

Поэтому приходится вводить специальную аксиому.

4). $\exists y\ \forall z\ \neg(z \in y)$
(Аксиома пустого множества)

Введем специальный символ $\varnothing$ для обозначения пустого множества, а запись $y = \varnothing$ будем рассматривать как сокращенное обозначение формулы

$$\forall z\ \neg(z \in y)\ .$$

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

3. Существует единственное объединение двух множеств

Казалось бы, объединение множеств легко ввести так же, как это было сделано для пересечения:

$$\forall x_1, x_2\ \exists y\ \forall z\ (z \in y\ \equiv\ (z \in x_1\ \vee\ z \in x_2))\ .$$

Но эта формула не подпадает под схему аксиом выделения

$$\forall x \forall u_1, \ldots, u_n\ \exists y\ \forall z\ (z \in y\ \equiv\ (z \in x\ \&\ \varphi(z, u_1, \ldots, u_n)))\ .$$

Можно было бы записать определение объединения так:

$$\forall X, x_1, x_2\ \exists y\ \forall z\ (z \in y\ \equiv\ (z \in X\ \&\ (z \neq x_1\ \vee\ z \in x_2)))\ .$$

Но совершенно непонятно, откуда взять подходящее множество X. Может быть, в качестве X взять $x_1 \cup x_2$? Но мы ведь еще не

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

Поэтому приходится вводить две специальные аксиомы.

5). $\forall y, z\ \exists x\ \forall u\ (u \in x\ \equiv\ (u = y\ \vee\ u = z))$
(Аксиома пары)

Множество x, существование которого утверждает аксиома пары, традиционно обозначается $\{y, z\}$.

6). $\forall y\ \exists x\ \forall u\ (u \in x\ \equiv\ \exists z\ (z \in y\ \&\ u \in z))$
(Аксиома объединения)

Множество x, существование которого утверждает аксиома объединения, традиционно обозначается $\bigcup\limits_{z \in y} z$ или более коротко $\cup y$. Таким образом $x_1 \cup x_2$ — это $\cup\{x_1, x_2\}$.

Теория множеств Цермело–Френкеля

Примеры применения аксиомы выделения

4. А что делать, если нам нужно множество, состоящее из одного-единственного элемента?

Для этого достаточно выделения и аксиомы пары:
множество, состоящее из одного элемента u — это множество $\{u,u\}$.

5. А что делать, если нам нужны упорядоченные наборы элементов?

Для этого достаточно аксиомы выделения и аксиомы пары:
упорядоченная пара $\langle y, z\rangle$ — это множество $\{y, \{y, z\}\}$.

Далее аналогично можно определять упорядоченные наборы (кортежи), функции, инъективные отображения, биективные отображения, отношения включения, равномощности и т. д.

Теория множеств Цермело–Френкеля

Но таким образом из пустого множества $\varnothing$, — единственного множества, существование которого гарантируют аксиомы, — можно получить только конечные множества. А откуда возьмутся бесконечные множества?

7). $\exists x\ (\varnothing \in x\ \&\ \forall y\ (y \in x\ \rightarrow\ y \cup \{y\} \in x))$

(Аксиома бесконечности)

Фактически, аксиома бесконечности определяет множество натуральных чисел:

$$\Big\{ \underbrace{\varnothing}_{0},\ \underbrace{\{\varnothing\}}_{1},\ \underbrace{\{\varnothing, \{\varnothing\}\}}_{2},\ \underbrace{\{\varnothing, \{\varnothing\}, \{\varnothing, \{\varnothing\}\}\}}_{3},\ \ldots \Big\}$$

Теория множеств Цермело–Френкеля

А откуда возьмутся несчетные множества?

8). $\forall y\ \exists x\ \forall z\ (z \in x\ \equiv\ \forall u\ (u \in z\ \rightarrow\ u \in y))$
(Аксиома степени)

Аксиома степени определяет множество всех подмножеств заданного множества (множество-степень, powerset). Значит, множества могут нарастать неограниченно «высоко».

Теория множеств Цермело–Френкеля

А являются ли множествами образы множеств относительно заданных функций, определяемых при помощи формул логики предикатов?

9). $\forall x\ (\forall y, z, u\ (y \in x\ \&\ \varphi(y, z)\ \&\ \varphi(y, u)\ \to z = u)\ \to$
$\to\ \exists v\ \forall w\ (w \in v\ \equiv\ \exists t\ (t \in x\ \&\ \varphi(t, w))))$
(Схема аксиом замены)

Теория множеств Цермело–Френкеля

А насколько «глубоко» могут опускаться множества? Не могут ли у нас образовываться такие множества, которые входят в состав самих себя в качестве элементов?

10). $\forall x\ (x \neq \varnothing\ \rightarrow\ \exists y\ (y \in x\ \&\ x \cap y = \varnothing))$
(Аксиома фундирования (регулярности))

Эта аксиома играет роль предохранителя, оберегающего теорию множеств от парадоксов. Аксиома фундирования объявляет, что семейства множеств вида

$$\left\{X_1, X_2, X_3, \dots\right\},$$

у которых $X_2 \in X_1$, $X_3 \in X_2$, $\dots$, $X_{n+1} \in X_n$, $\cdots$ и т. д. множествами не являются .

Теория множеств Цермело–Френкеля

Примеры применения аксиомы фундирования

$ZF \vdash \forall u(u \notin u)$
Из аксиомы фундирования

$$\forall x\ (x \neq \varnothing\ \rightarrow\ \exists y\ (y \in x\ \&\ x \cap y = \varnothing))$$

следует (если в качестве x выбрать $\{u\}$)

$$ZF \vdash \exists y\ (y \in \{u\}\ \&\ \{u\} \cap y = \varnothing)\ .$$

Поскольку единственным элементом y в множестве $\{u\}$ является u, получаем

$$ZF \vdash \{u\} \cap u = \varnothing\ .$$

Следовательно, $ZF \vdash u \notin u$.

Теория множеств Цермело–Френкеля

Примеры применения аксиомы фундирования

Попробуйте самостоятельно убедиться, что из аксиом теории множеств Цермело–Френкеля следует невозможность существования «парадоксальных множеств»:

$$ZF \vdash \forall u, v\ (u \notin v\ \vee v \notin u)$$

$$ZF \vdash \neg\exists x\ \forall y\ (y \in x\ \equiv y \notin y)$$

Нужны ли еще какие-нибудь другие аксиомы?

Теория множеств Цермело–Френкеля

К сожалению, для решения некоторых задач приходится вводить дополнительные аксиомы.

Например, интуиция подсказывает, что любые два множества должны быть сравнимы по мощности. Два множества A и B называются равномощными ($A \sim B$), если существует биективная функция, отображающая одно множество на другое. Справедливо ли следующее утверждение?

Теорема трихотомии. Для любых двух множеств A и B верно одно из трех:

- либо $A \sim B$,
- либо $A \not\sim B$, но существует такое A', $A' \subset A$, что $A' \sim B$,
- либо $A \not\sim B$, но существует такое B', $B' \subset B$, что $A \sim B'$.

Эту теорему можно доказать, но лишь при том условии, если у нас есть хоть какой-нибудь способ, позволяющий выбрать из произвольного непустого множества хоть какой-нибудь элемент. Чтобы этот способ выбора стал легальным средством доказательства, нужно ввести специальную аксиому выбора.

Теория множеств Цермело–Френкеля

Аксиома выбора (CA)

Каково бы ни было множество попарно непересекающихся множеств $U = \{X_1, X_2, \dots\}$, существует множество Y, содержащее в точности по одному представителю из каждого множества $X_1, X_2, \dots$ семейства U.

Аксиома выбора используется при доказательстве очень большого числа теорем математики. С ее помощью можно доказать весьма неожиданные утверждения. К их числу относится

Теорема Цермело

Любое множество можно вполне упорядочить, т. е. определить на этом множестве такое отношение линейного порядка, при котором не существует бесконечно убывающих последовательностей элементов.

Теория множеств Цермело–Френкеля

А не привнесет ли аксиома выбора какое-нибудь противоречие в теорию ZF? Этот вопрос остается открытым и по сей день.

Есть в теории множеств и другие задачи, для решения которых недостаточно аксиом теории множеств ZF.

Континуум-гипотеза (CH)

Любое подмножество множества вещественных чисел либо является счетным, либо равномощно множеству вещественных чисел (является континуальным).

В 1939 г. К. Гедель доказал теорему:

Если теория множеств ZF+CA непротиворечива, то теория ZF+AC+CH также непротиворечива .

В 1963 г. П. Коэн доказал теорему:

Если теория множеств ZF+CA непротиворечива, то теория ZF+AC+$\neg$CH также непротиворечива .

Формальная арифметика

А можно ли полностью аксиоматизировать арифметику натуральных чисел?

В 1889 г. итальянский математик Д. Пеано предложил список аксиом, при помощи которых можно доказывать утверждения о свойствах натуральных чисел.

Арифметика Пеано (PA) образуется за счет добавления к КИП$_=$ сигнатуры $\langle 0, s, +, \times \rangle$ следующих аксиом.
Здесь $s(x)$ нужно рассматривать как одноместную операцию, реализующую функцию вычисления следующего натурального числа $x + 1$.

Формальная арифметика

1. $\forall x, y\ (s(x) = s(y)\ \rightarrow x = y)$;
2. $\forall x\ (s(x) \neq 0)$;
3. $\forall x\ \exists y(x \neq 0\ \rightarrow\ x = s(y))$;
4. $\forall x\ (x + 0 = x)$;
5. $\forall x, y\ (x + s(y) = s(x + y))$;
6. $\forall x\ (x \times 0 = 0)$;
7. $\forall x, y\ (x \times s(y) = x \times y + x)$;
8. $\varphi(0)\ \&\ \forall x\ (\varphi(x)\ \rightarrow\ \varphi(s(x)))\ \rightarrow\ \forall x\ \varphi(x)$.

Вопрос о непротиворечивости и полноте этой аксиоматической теории долгое время оставался центральной проблемой математики. В 1931 г. К. Гедель доказал теорему, которая дала совершенно неожиданный ответ на этот вопрос.

Формальная арифметика

Нумералы и арифметизуемые отношения

Нумералом $\bar{\mathbf{n}}$ натурального числа n называется терм

$$\underbrace{s(s(\ldots s(}_{n \text{ раз}} 0)\ldots))$$

Например, $\bar{\mathbf{4}}$ — это терм $s(s(s(s(0))))$.

Отношение $P^{(k)}$ на множестве натуральных чисел называется **арифметизуемым**, если существует такая формула $\varphi(x_1, x_2, \ldots, x_k)$, что для всякого набора натуральных чисел $(n_1, n_2, \ldots, n_k)$ верны соотношения

- $P^{(k)}(n_1, n_2, \ldots, n_k) = \mathit{true} \iff PA \vdash \varphi(\bar{\mathbf{n}}_1, \bar{\mathbf{n}}_2, \ldots, \bar{\mathbf{n}}_k)$,
- $P^{(k)}(n_1, n_2, \ldots, n_k) = \mathit{false} \iff PA \vdash \neg\varphi(\bar{\mathbf{n}}_1, \bar{\mathbf{n}}_2, \ldots, \bar{\mathbf{n}}_k)$.

Формальная арифметика

Нумералы и арифметизуемые отношения

Теорема Геделя–Тьюринга.
Отношение $P^{(k)}$ на множестве натуральных чисел арифметизуемо в том и только том случае, если существует такая машина Тьюринга M, которая для любого набора натуральных чисел $(n_1, n_2, \ldots, n_k)$ имеет завершающееся вычисление, преобразующее начальную конфигурацию

$$q_1 \underbrace{11\ldots1}_{n_1+1 \text{ раз}} 0 \underbrace{11\ldots1}_{n_2+1 \text{ раз}} 0\ldots0 \underbrace{11\ldots1}_{n_k+1 \text{ раз}}$$

- в заключительную конфигурацию $q_0 1$, если $P^{(k)}(n_1, n_2, \ldots, n_k) = \mathit{true}$,
- в заключительную конфигурацию $q_0 0$, если $P^{(k)}(n_1, n_2, \ldots, n_k) = \mathit{false}$.

Формальная арифметика

Нумерация Геделя

Закодируем натуральными числами (занумеруем) символы алфавита формальной арифметики, формулы и конечные последовательности формул.

$gn(0) = 3,\ gn(s) = 5,\ gn(+) = 7,\ gn(\times) = 9,\ gn(=) = 11,$

$gn(\neg) = 13,\ gn(\&) = 15,\ gn(\vee) = 17,\ gn(\to) = 19,$
$gn(\forall) = 21,\ gn(\exists) = 23,$

$gn(\ (\ ) = 25,\ gn(\ )\ ) = 27,$

$gn(x_1) = 29,\ gn(x_2) = 31,\ \ldots,\ gn(x_i) = 27 + 2i, \ldots.$

Геделев номер слова:
$gn(a_1 a_2 a_3 \ldots a_n) = 2^{gn(a_1)} 3^{gn(a_2)} 5^{gn(a_3)} \ldots p_n^{gn(a_n)}.$

Геделев номер последовательности слов:
$gn(\alpha_1 \alpha_2 \alpha_3 \ldots \alpha_m) = 2^{gn(\alpha_1)} 3^{gn(\alpha_2)} 5^{gn(\alpha_3} \ldots p_m^{gn(\alpha_m)}.$

Формальная арифметика

Примеры арифметизуемых отношений

Рассмотрим два отношения

1. $\mathrm{Form}^{(1)}$: $\mathrm{Form}(n) = \mathit{true} \iff n$ — геделев номер формулы арифметики Пеано.
2. $\mathrm{Proof}^{(2)}$: $\mathrm{Proof}(n, m) = \mathit{true} \iff n$ — геделев номер некоторой формулы φ арифметики Пеано, а m — геделев номер конечной последовательности формул, составляющей доказательство формулы φ.

Лемма

Отношения Form и Proof арифметизируемы.

Обозначим *Proof* арифметическую формулу, реализующую предикат Proof.

Формальная арифметика

Странные предикаты

Ну, если вы поверили, что предикат $\mathrm{Proof}^{(2)}$ арифметизуем, то совершенно очевидно, что арифметизуемым является и такой странный предикат $\mathrm{MetaProof}^{(2)}$:

$$\mathrm{MetaProof}(n, m) = \mathit{true}$$

$$\Updownarrow$$

n — геделев номер некоторой формулы арифметики Пеано, $\varphi(x)$, зависящей от одной переменной,
а m — геделев номер конечной последовательности формул, составляющей доказательство формулы $\varphi(\bar{\mathbf{n}})$.

Но если предикат $\mathrm{MetaProof}^{(2)}$ арифметизуем, то существует арифметическая формула $\mathcal{W}(x, y)$, выражающая отношение $\mathrm{MetaProof}$.

Формальная арифметика

Странные предикаты

Рассмотрим формулу $\varphi(x) = \neg\exists y\ \mathcal{W}(x, y)$ и ее геделев номер $n_0 = gn(\varphi(x))$.

Интересно, а что за высказывание выражает замкнутая формула $\varphi(\bar{\mathbf{n}}_0)$?

Это высказывание таково: Нельзя доказать формулу $\varphi(\bar{\mathbf{n}}_0)$, т. е. формула $\varphi(\bar{\mathbf{n}}_0)$ утверждает, что она недоказуема.

Таким образом, мы имеем дело со строго сформулированным аналогом «парадокса лжеца».

И если эта формула действительно не имеет доказательства в арифметике Пеано, то она выражает истинное суждение.

Теорема Геделя о неполноте PA

(облегченный вариант)

Если множество натуральных чисел с операциями сложения и умножения $(\mathcal{N}_0, +, \times)$ является моделью для аксиом PA, то PA неполна.

Доказательство.

1. Покажем, что $PA \nvdash \varphi(\bar{\mathbf{n}}_\mathbf{0})$.
Допустим противное $PA \vdash \varphi(\bar{\mathbf{n}}_\mathbf{0})$. Тогда формула $\varphi(\bar{\mathbf{n}}_\mathbf{0})$ имеет доказательство в PA: $\psi_1, \psi_2, \ldots, \psi_N = \varphi(\bar{\mathbf{n}}_\mathbf{0})$.

Пусть $m = gn(\psi_1, \psi_2, \ldots, \psi_N)$. Тогда $MetaProof(n_0, m) = true$.
Поэтому, учитывая арифметизуемость предиката $MetaProof$, получаем $PA \vdash \mathcal{W}(\bar{\mathbf{n}}_\mathbf{0}, \bar{\mathbf{m}})$. Но это означает, что $PA \vdash \exists y\ \mathcal{W}(\bar{\mathbf{n}}_\mathbf{0}, y)$ и, следовательно, $PA \vdash \neg\varphi(\bar{\mathbf{n}}_\mathbf{0})$.
Но это означает, что PA — противоречивая теория, вопреки условию теоремы (PA имеет модель).

Теорема Геделя о неполноте PA

(облегченный вариант)

Если множество натуральных чисел с операциями сложения и умножения $(\mathcal{N}_0, +, \times)$ является моделью для аксиом PA, то PA неполна.

Доказательство.

2. Покажем, что $PA \nvdash \neg\varphi(\bar{\mathbf{n}}_\mathbf{0})$.
Допустим противное $PA \vdash \neg\varphi(\bar{\mathbf{n}}_\mathbf{0})$, т. е. $PA \vdash \exists y\ \mathcal{W}(\bar{\mathbf{n}}_\mathbf{0}, y)$.
Тогда (почему?) существует такое натуральное число m, для которого верно $PA \vdash \mathcal{W}(\bar{\mathbf{n}}_\mathbf{0}, \bar{\mathbf{m}})$. Учитывая, что формула $\mathcal{W}$ выражает отношение *MetaProof* , приходим к выводу: m — это геделев номер доказательства формулы $\varphi(\bar{\mathbf{n}}_\mathbf{0})$ в PA. Значит, $PA \vdash \varphi(\bar{\mathbf{n}}_\mathbf{0})$.
Но это означает, что PA — противоречивая теория, вопреки условию теоремы (PA имеет модель).

Теорема Геделя о неполноте PA

(облегченный вариант)

Если множество натуральных чисел с операциями сложения и умножения $(\mathcal{N}_0, +, \times)$ является моделью для аксиом PA, то PA неполна.

Доказательство.

3. Итак,

$PA \nvdash \varphi(\bar{\mathbf{n}}_\mathbf{0})$

$PA \nvdash \neg\varphi(\bar{\mathbf{n}}_\mathbf{0})$.

Значит, $\varphi(\bar{\mathbf{n}}_\mathbf{0}) = \neg\exists y\ \mathcal{W}(\bar{\mathbf{n}}_\mathbf{0}, y)$ — это истинное арифметическое утверждение, которое нельзя ни доказать, ни опровергнуть в арифметике Пеано.
Значит, арифметика Пеано неполна. $\square$

Теорема Геделя о неполноте PA

(Основной вариант)

Пусть запись *Consist* обозначает арифметическую формулу

$$\neg\exists X \; \mathit{Proof}(\overline{\mathit{gn}(0 = s(0))}, X)$$

Если формальная арифметика PA непротиворечива, то

$$\begin{aligned} PA &\nvdash \mathit{Consist} \\ PA &\nvdash \neg \mathit{Consist}. \end{aligned}$$

Это означает, что аксиоматические теории (сколь бы выразительны они ни были) не позволяют построить доказательство их собственной непротиворечивости.

Последовательности Гудштейна

Один из наиболее ярких примеров, показывающих неполноту аксиоматической арифметики, был предложен английским математиком Рубеном Гудштейном.

Каждое натуральное число m может быть представлено единственным образом в n-чной системе счисления

$$m = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_1 n^1 + a_0 n^0 \ ,$$

где $a_k, a_{k-1}, \ldots, a_1, a_0$ — числа из диапазона $0, \ldots, n-1$. Но степени $k, k-1, \ldots, 1, 0$ также можно представить в n-чной системе счисления. И в представлении этих степеней также можно соответствующие степени представлять в n-чной системе счисления.

И окончательное представление числа m, в котором все числа — и коэффициенты и степени — представлены в n-чной системе счисления, называется наследственно n-чной системой счисления. Такое представление будем обозначать $H(m, n)$.

Последовательности Гудштейна

Например,

$$\begin{aligned} 555 &= 1\cdot 2^9 + 1\cdot 2^5 + 1\cdot 2^3 + 1\cdot 2^1 + 1\cdot 2^0 \\ &= 1\cdot 2^{1\cdot 2^3+1\cdot 2^0} + 1\cdot 2^{1\cdot 2^2+1\cdot 2^0} + 1\cdot 2^{1\cdot 2^1+1\cdot 2^0} + 1\cdot 2^1 + 1\cdot 2^0 \\ &= 1\cdot 2^{1\cdot 2^{1\cdot 2^1+1}+1\cdot 2^0} + 1\cdot 2^{1\cdot 2^2+1\cdot 2^0} + 1\cdot 2^{1\cdot 2^1+1\cdot 2^0} + 1\cdot 2^1 + 1\cdot 2^0. \end{aligned}$$

Таким образом,

$$H(555,2) = 1\cdot 2^{1\cdot 2^{1\cdot 2^1+1}+1} + 1\cdot 2^{1\cdot 2^2+1} + 1\cdot 2^{1\cdot 2^1+1} + 1\cdot 2^1 + 1.$$

Последовательности Гудштейна

Последовательность Гудштейна $G(N)$ определяется так:

1. Возьмите произвольное натуральное число N. Это число — первый член последовательности a_1.
2. Чтобы получить второй член a_2, постройте $H(a_1, 2)$, замените в этом представлении 2 на 3 и вычтите из получившегося числа 1.

n. Чтобы получить n-й член a_n, постройте $H(a_{n-1}, n)$, замените в этом представлении n на $n+1$ и вычтите из получившегося числа 1.

Теорема Гудштейна о сходимости $G(N)$.

Любая последовательность Гудштейна сходится к 0, т.е. для любого натурального числа N существует такое натуральное число n, для которого $a_n = 0$.

Последовательности Гудштейна

Например, последовательность $G(3)$ сходится к 0 через 6 шагов:

$$3, 3, 3, 2, 1, 0$$

Последовательность $G(4)$ сходится спустя $3 \cdot 2^{402653211} - 1$ шагов.

Теорема Кирби-Пари

Утверждение о сходимости любой последовательности $G(N)$ недоказуемо в арифметике Пеано.

КОНЕЦ ЛЕКЦИИ 23