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

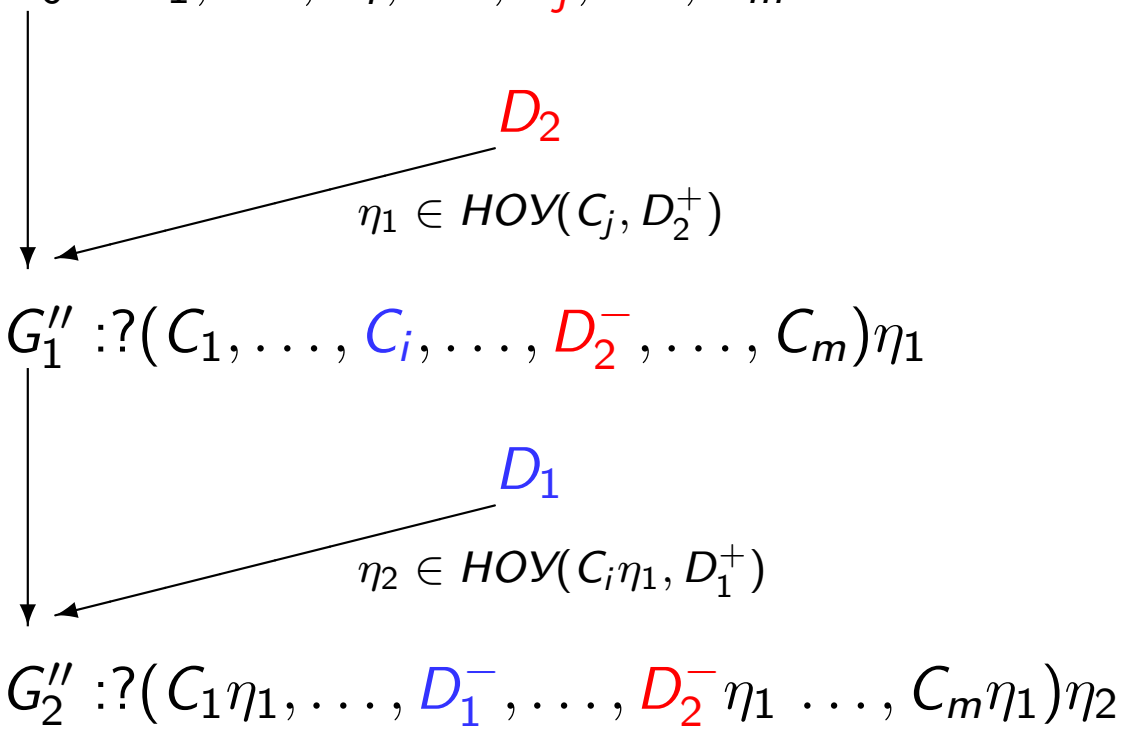
$\theta_2 \in HOV(C_j\theta_1, D_2^+)$

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

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

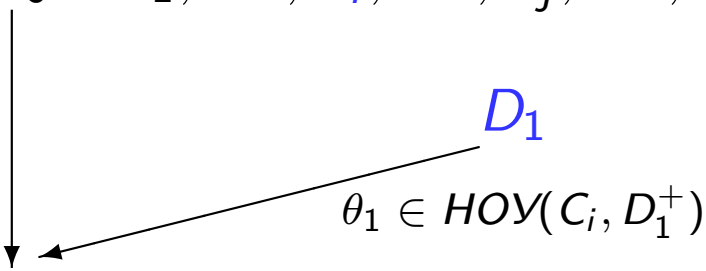
Переключательная лемма говорит о том, что при изменении порядка выбора подцелей результат вычисления сохраняется (с точностью до переименования переменных).

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

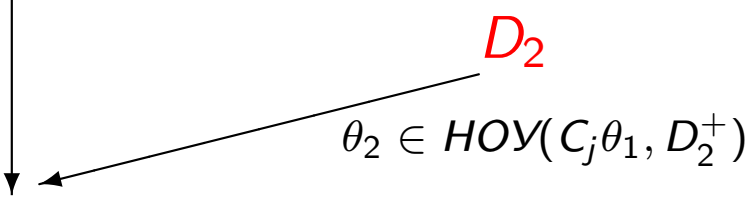
Доказательство (переключательной леммы).

Рассмотрим первое вычисление

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



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

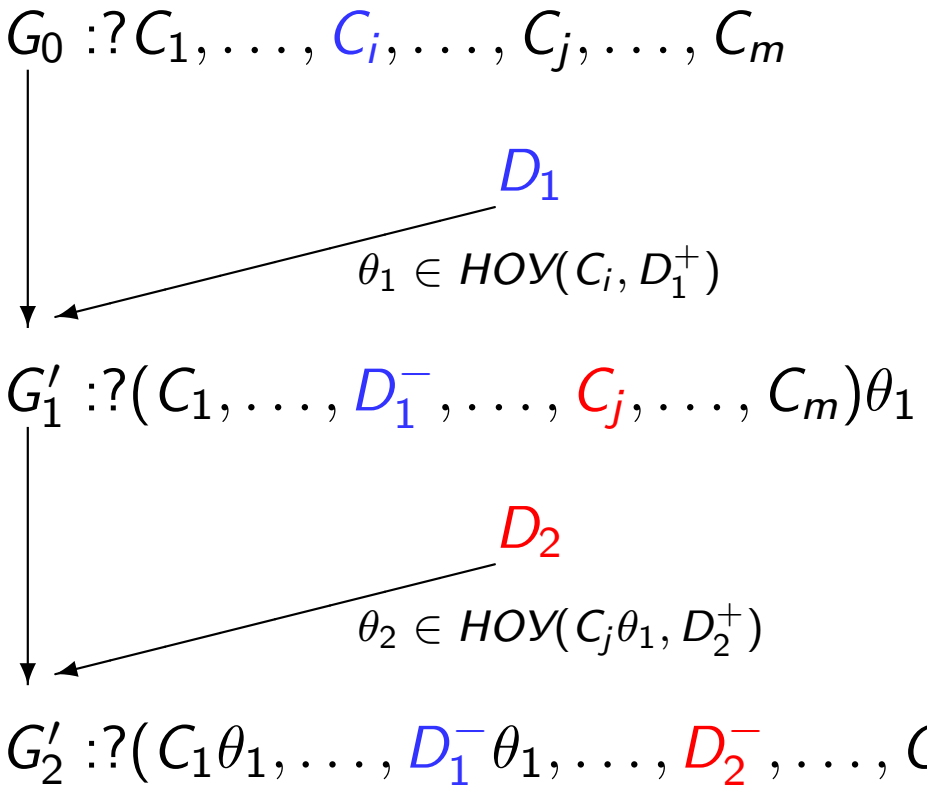


$G_2' :? (C_1\theta_1, \ldots, D_1^-\theta_1, \ldots, D_2^-, \ldots, C_m\theta_1)\theta_2$

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 $Dom_{\theta_1} \cap Var_{D_2} = \emptyset$.

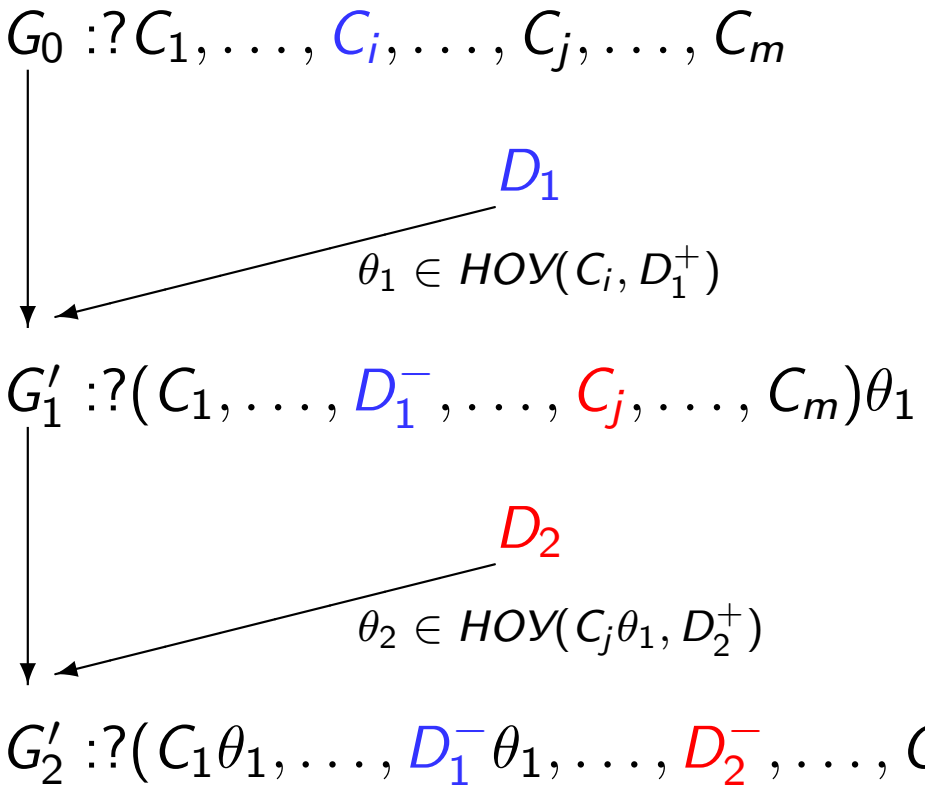
Поэтому $D_2^+\theta_2 = D_2^+\theta_1\theta_2$.

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

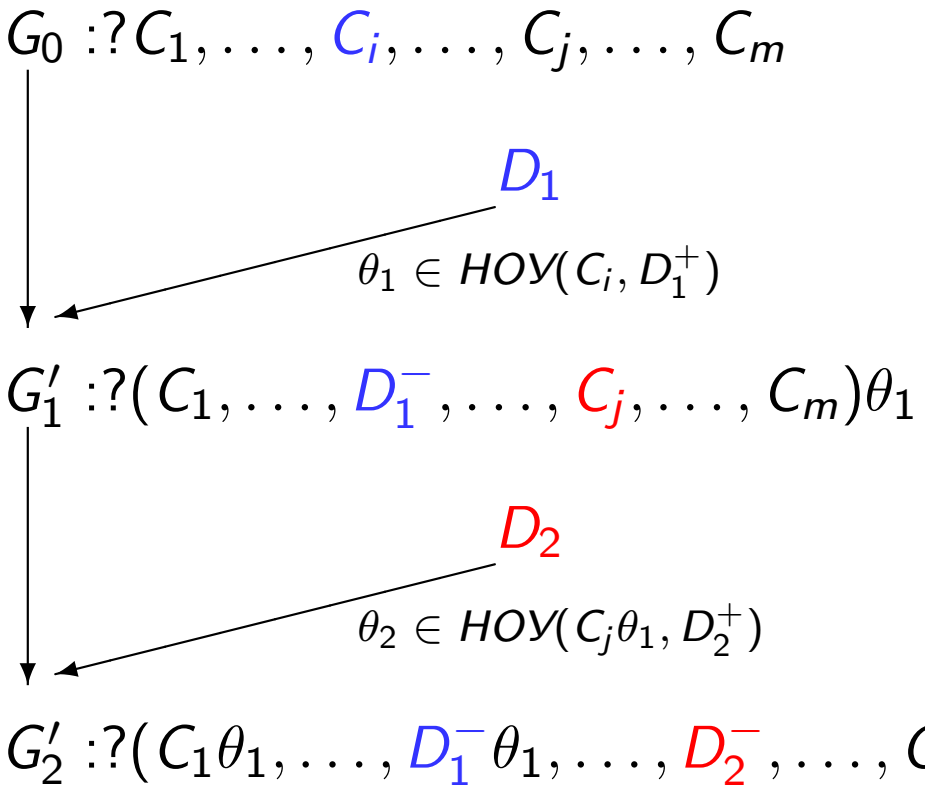


А раз C_j и D_2^+ унифицируемы, существует $\eta_1 \in HOY(C_j, D_2^+)$, и при этом $\theta_1\theta_2 = \eta_1\lambda$.

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

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

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

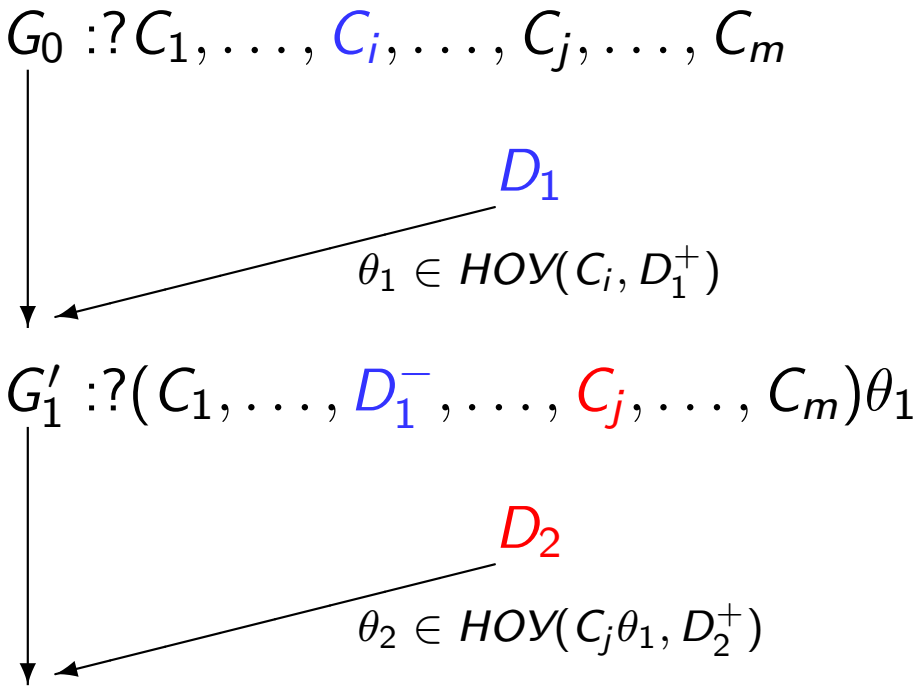
Далее, заметим,
что $C_i\theta_1\theta_2 = D_1^+\theta_1\theta_2$,
и поэтому $C_i\eta_1\lambda = D_1^+\eta_1\lambda$.

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 $C_i\eta_1\lambda = D_1^+\eta_1\lambda$.

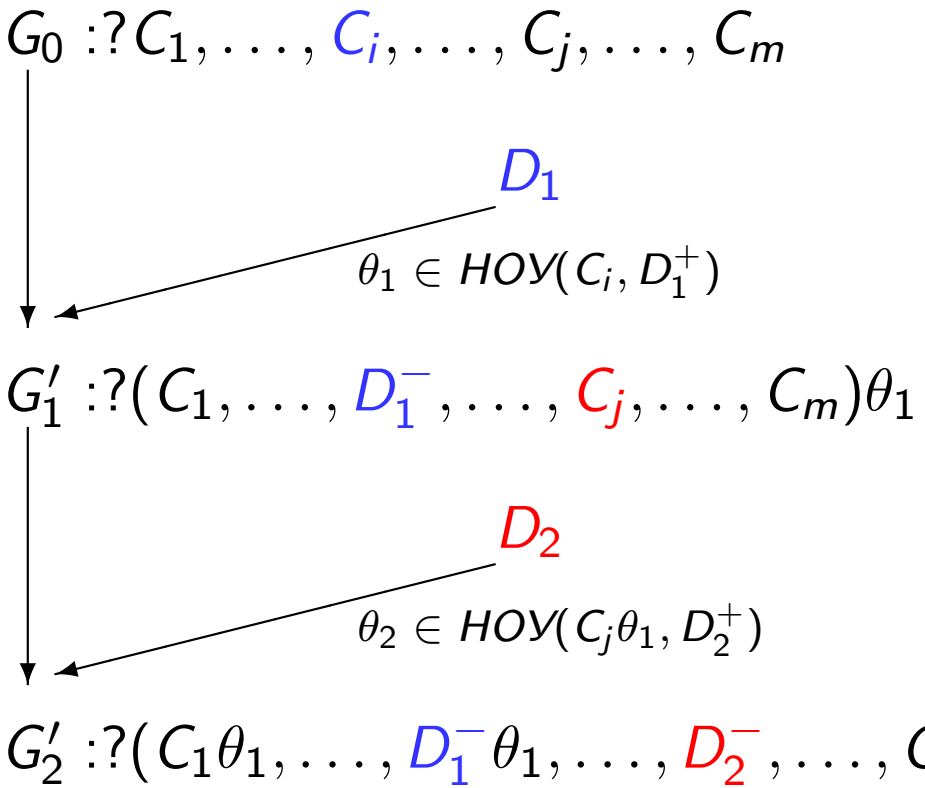
Т. к. $Dom_{\eta_1} \cap Var_{D_1} = \emptyset$,
верно $D_1^+\eta_1 = D_1^+$.

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

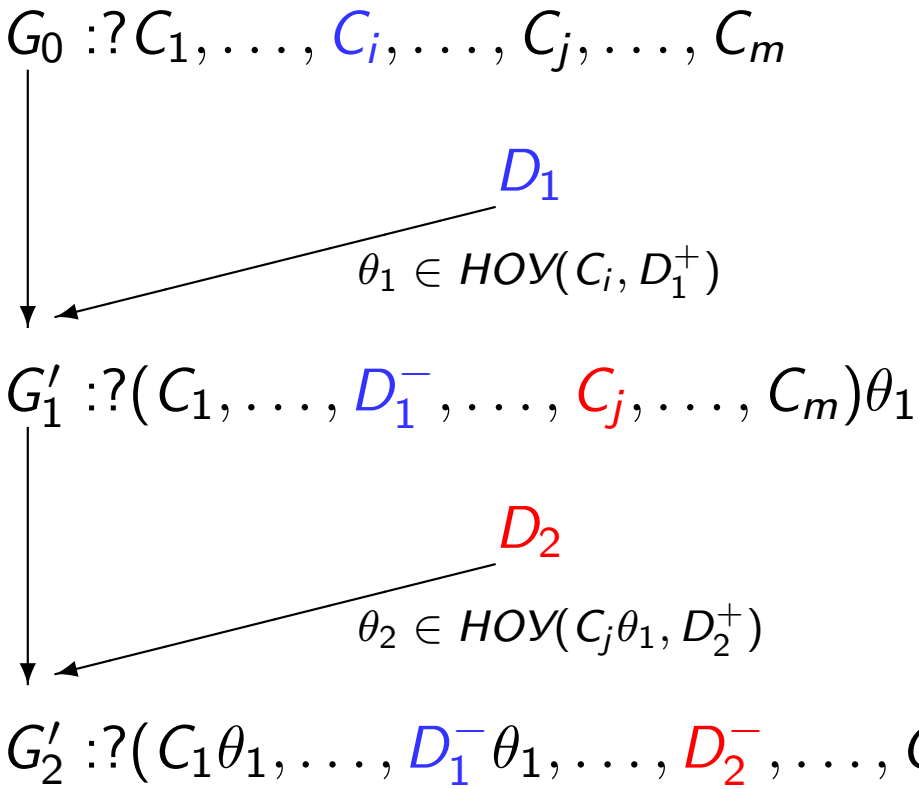


А раз $C_i\eta_1$ и D_1^+ унифицируемы, существует $\eta_2 \in HOV(C_i\eta_1, D_1^+)$, и при этом $\lambda = \eta_2\rho'$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

Доказательство (переключательной леммы).

Рассмотрим первое вычисление



Итак, получаем

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

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

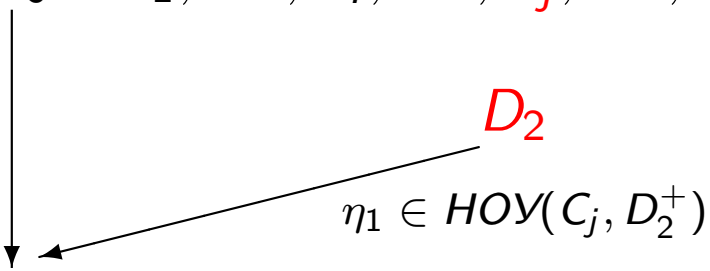
$\theta_1\theta_2 = \eta_1\lambda = \eta_1\eta_2\rho'$.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

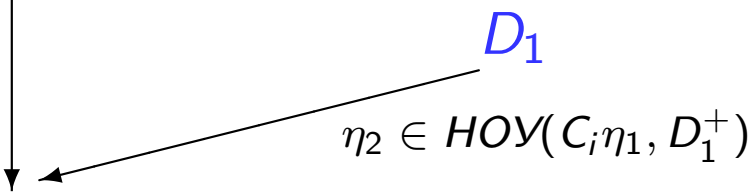
Доказательство (переключательной леммы).

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

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

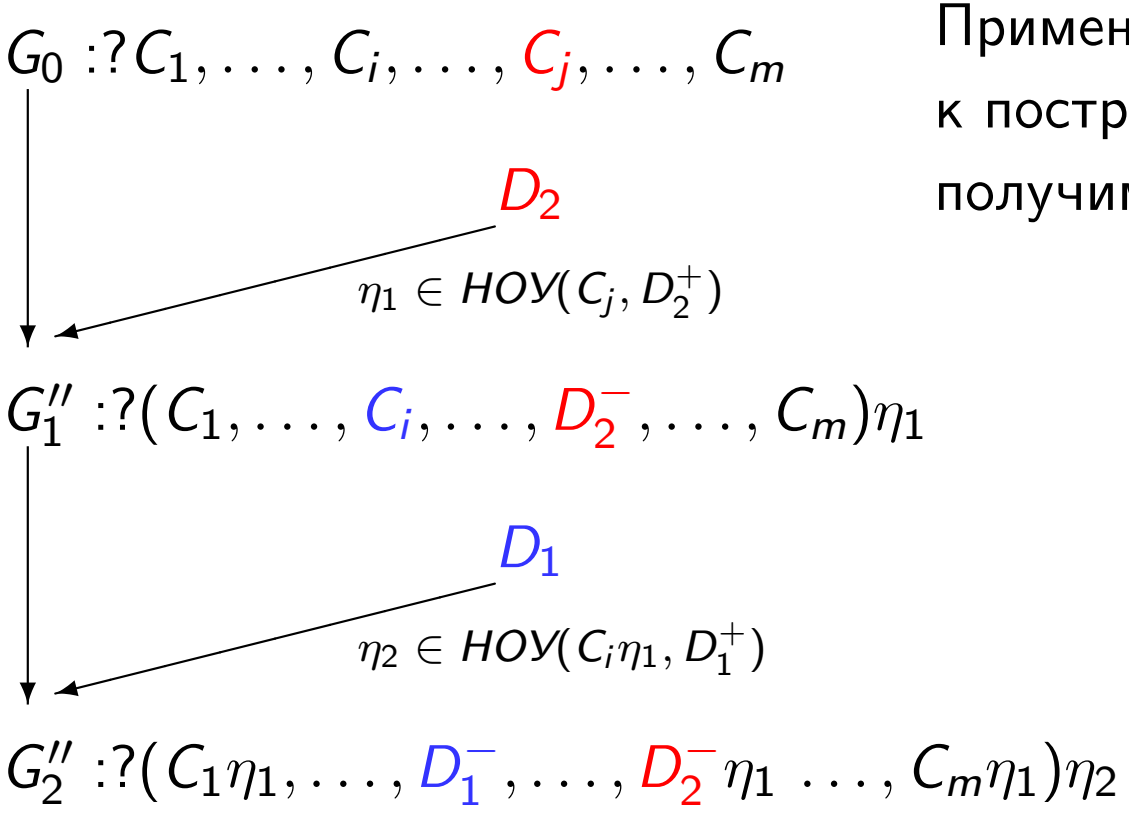
$\eta_2 \in HOY(C_i\eta_1, D_1^+)$,

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

Значит, запрос G_0 имеет и другое вычисление.

ПРАВИЛА ВЫБОРА ПОДЦЕЛЕЙ

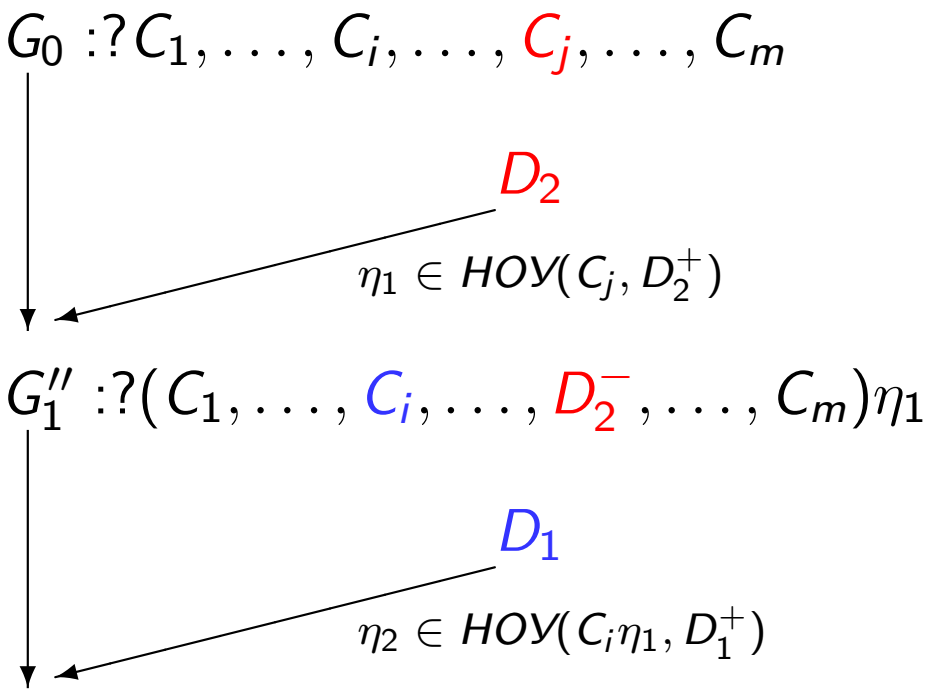
Доказательство (переключательной леммы).



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

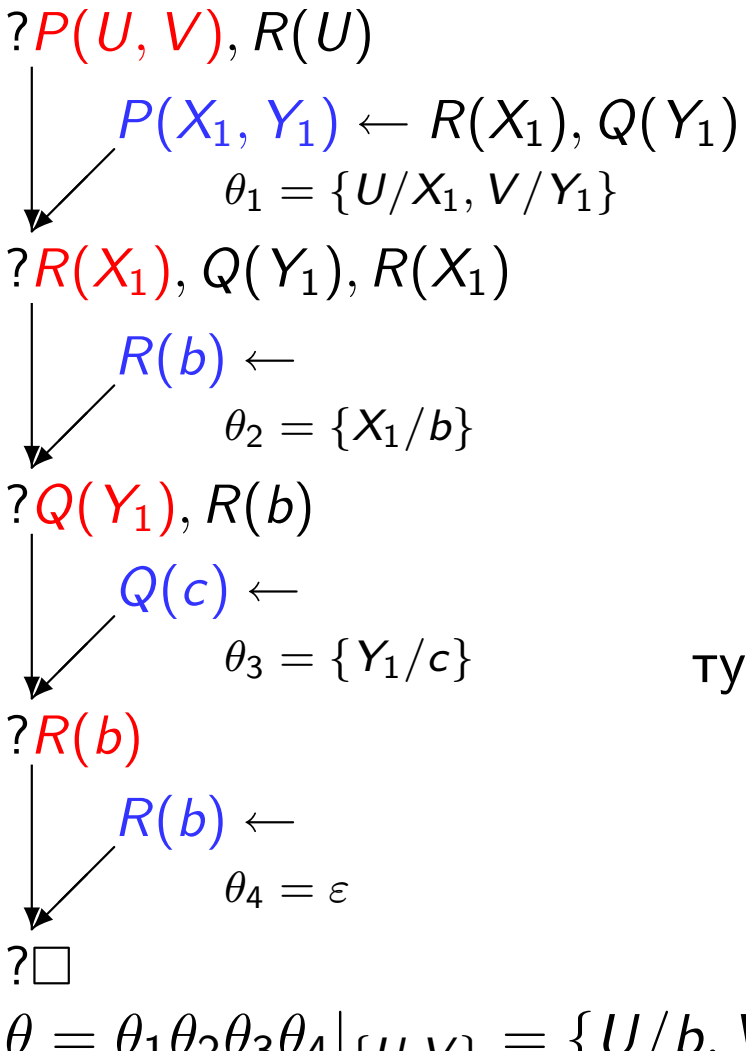
Обратимся снова к запросу $?P(U, V), R(U)$ к логической программе

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

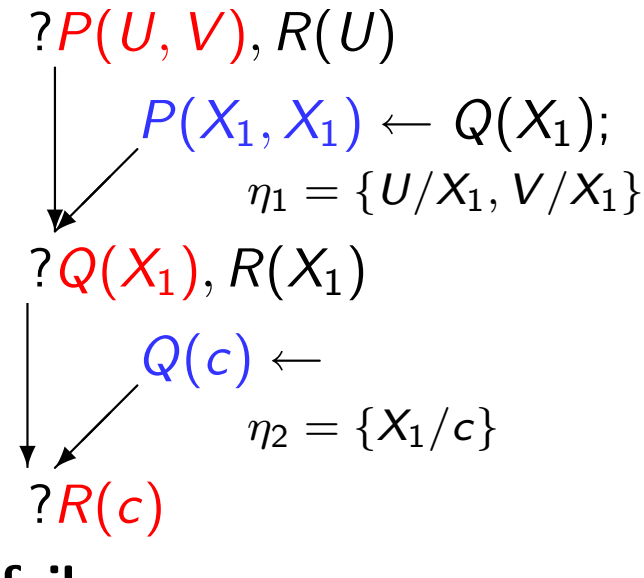
и будем применять стандартное правило выбора подцелей.

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

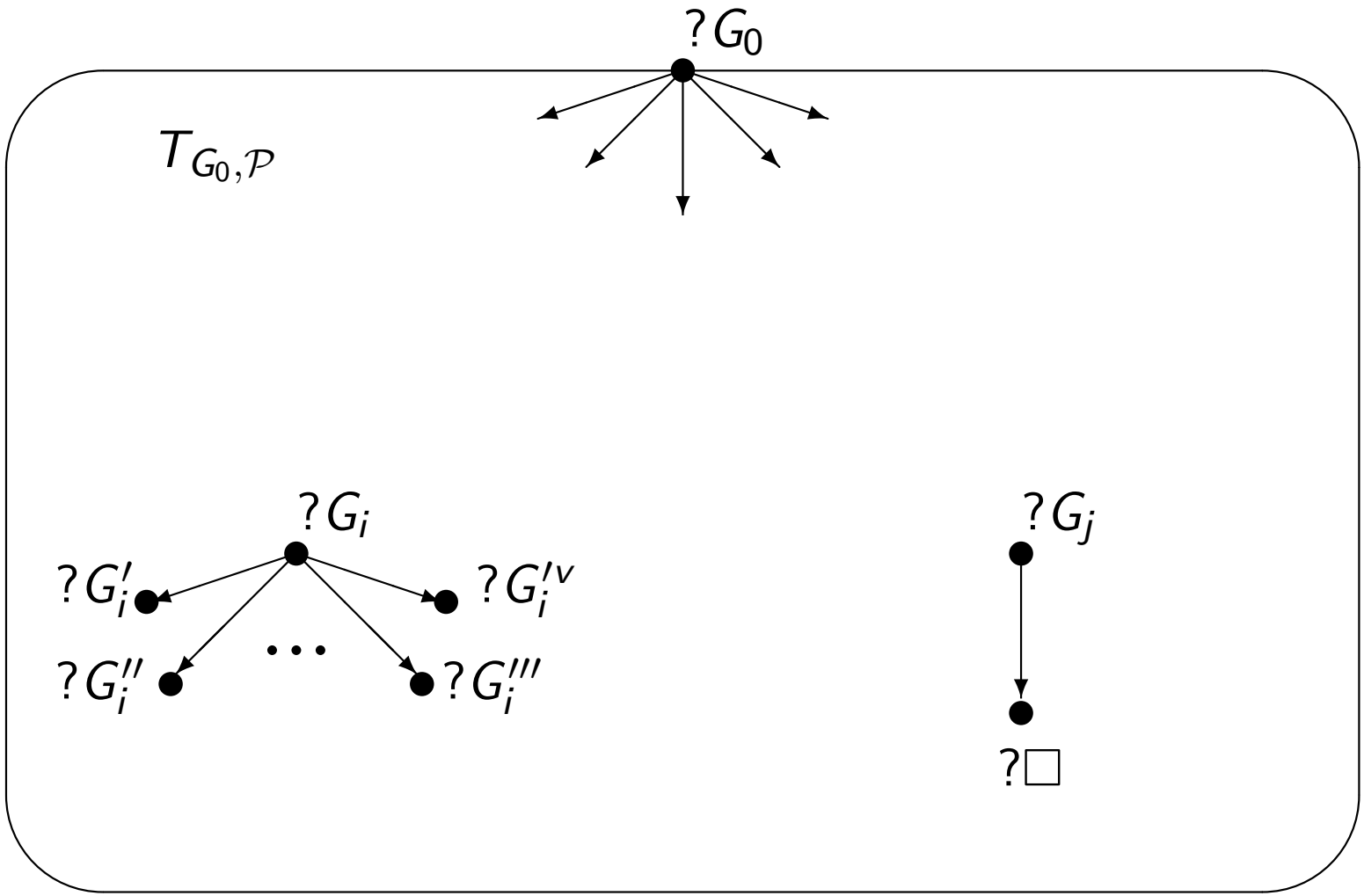
Определение

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

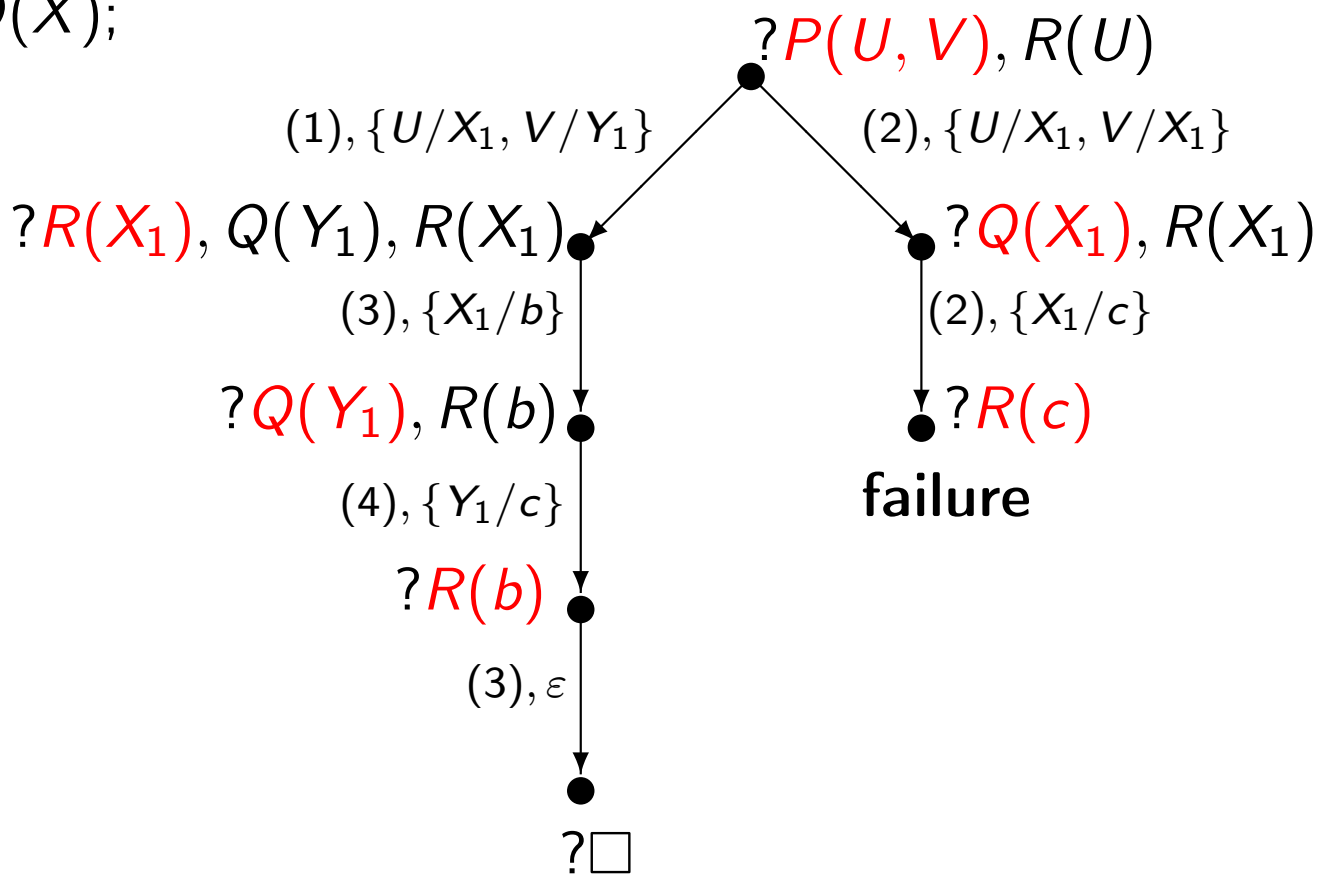
Иллюстрация



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

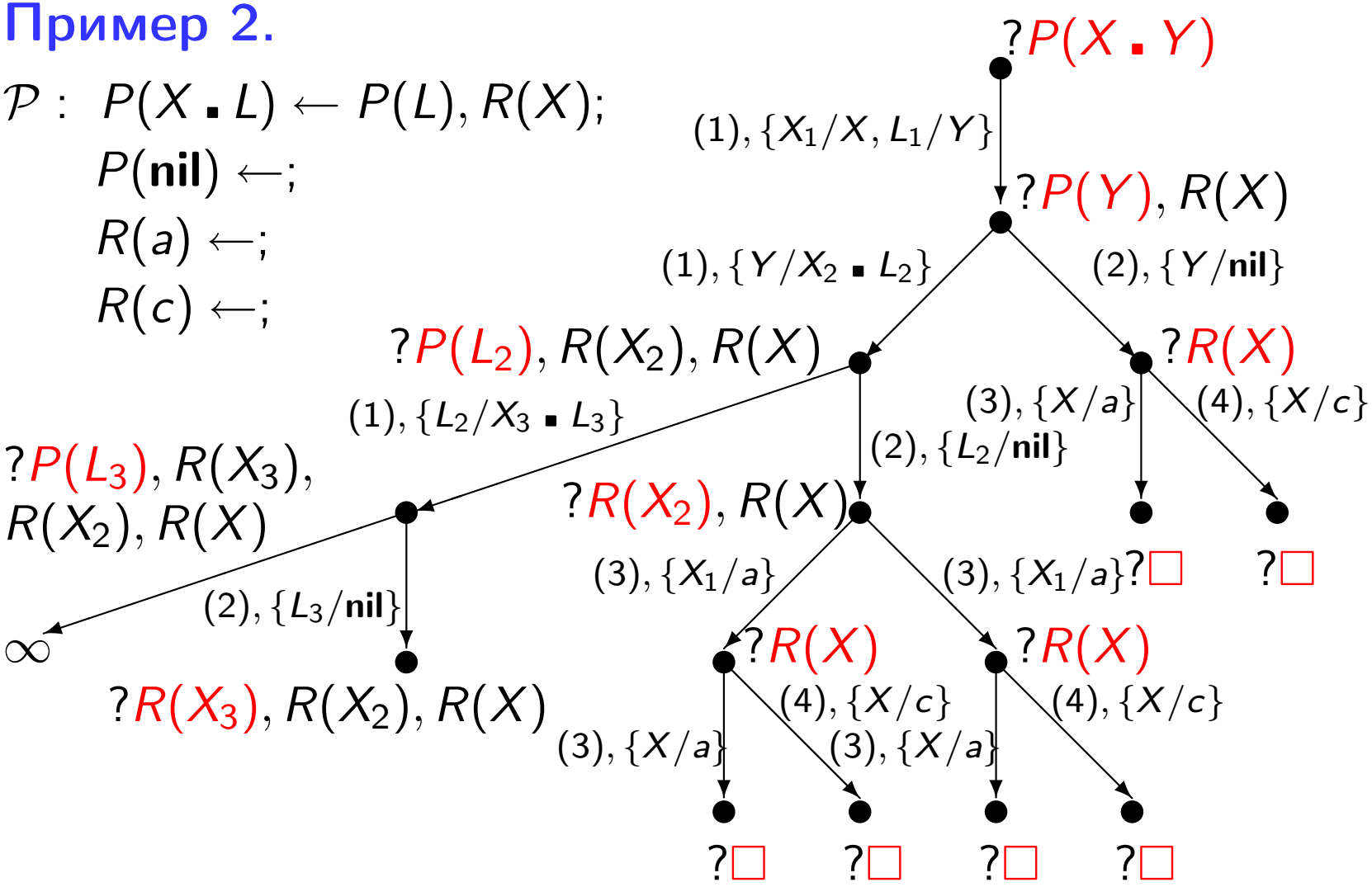
Пример 2.

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

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

$R(a) \leftarrow;$

$R(c) \leftarrow;$



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

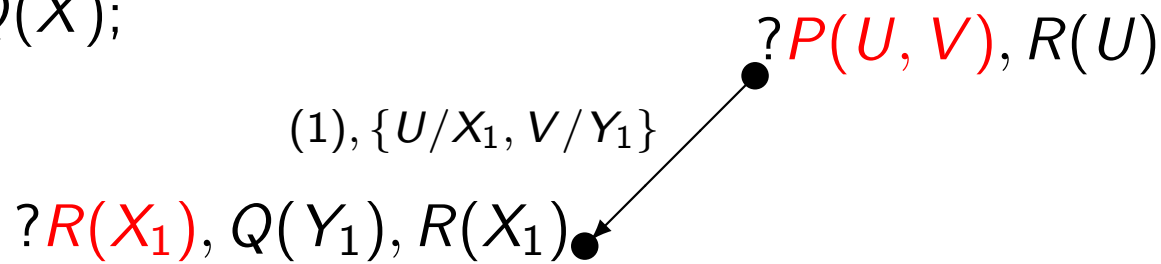
Пример обхода в ширину.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

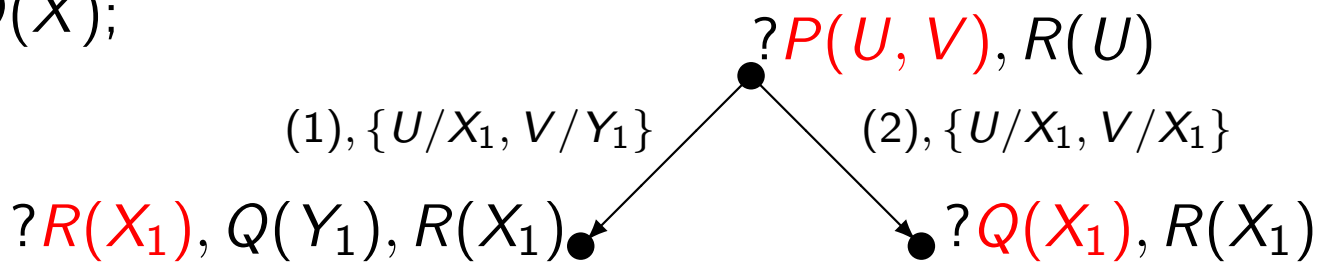
Пример обхода в ширину.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

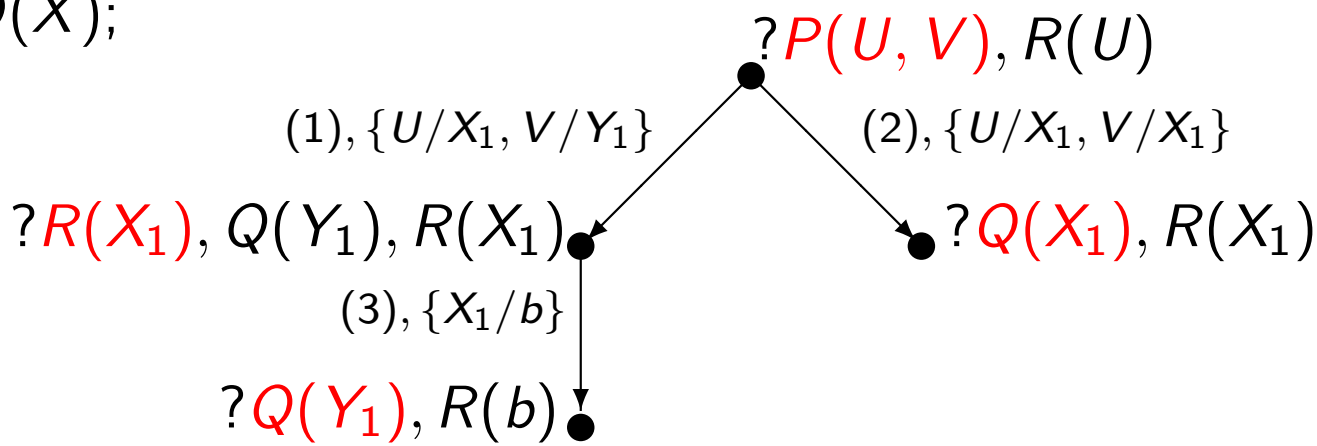
Пример обхода в ширину.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

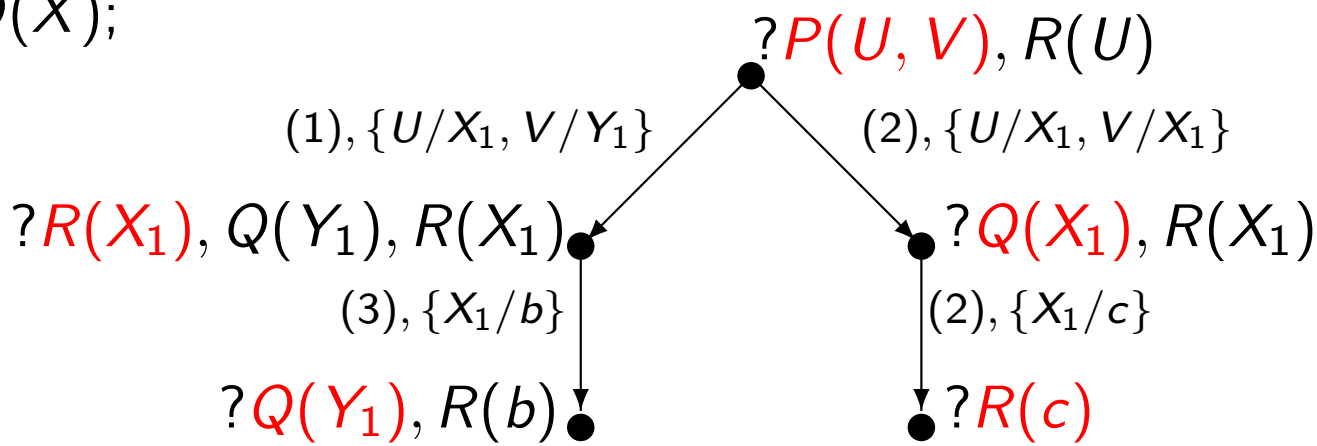
Пример обхода в ширину.

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

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

$R(b) \leftarrow;$

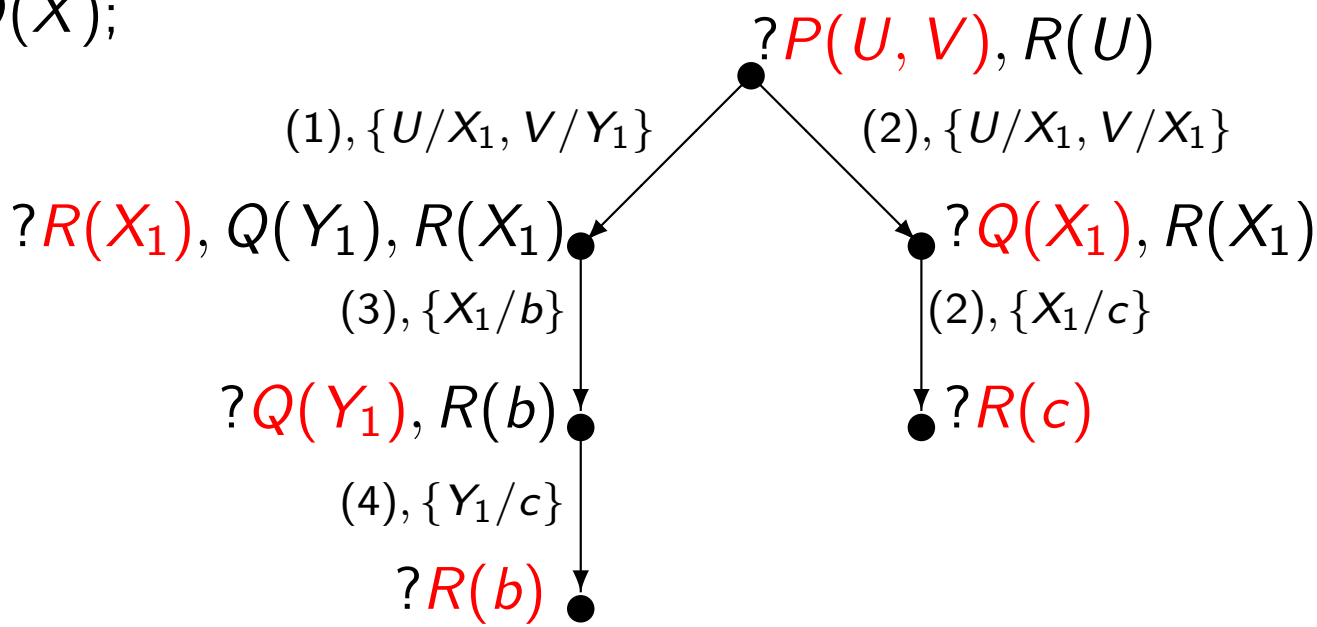
$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

Пример обхода в ширину.

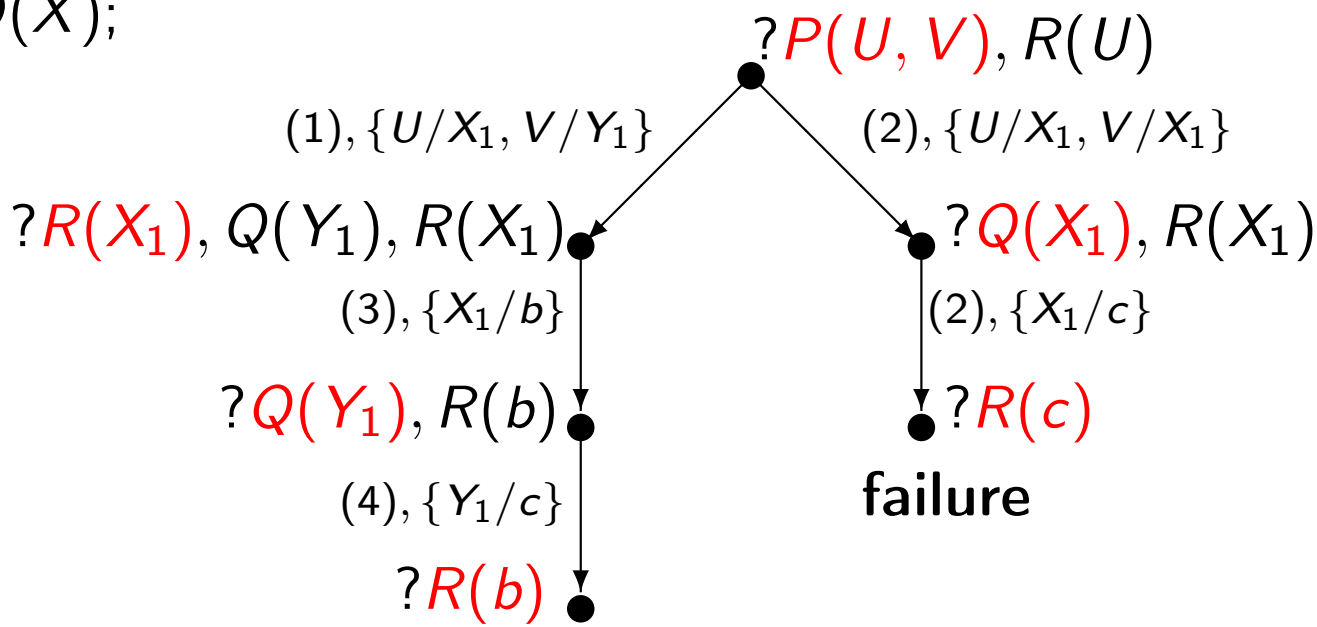
$\mathcal{P}:\ P(X,Y) \leftarrow R(X), Q(Y);$
$P(X,X) \leftarrow Q(X);$
$R(b) \leftarrow;$
$Q(c) \leftarrow;$



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

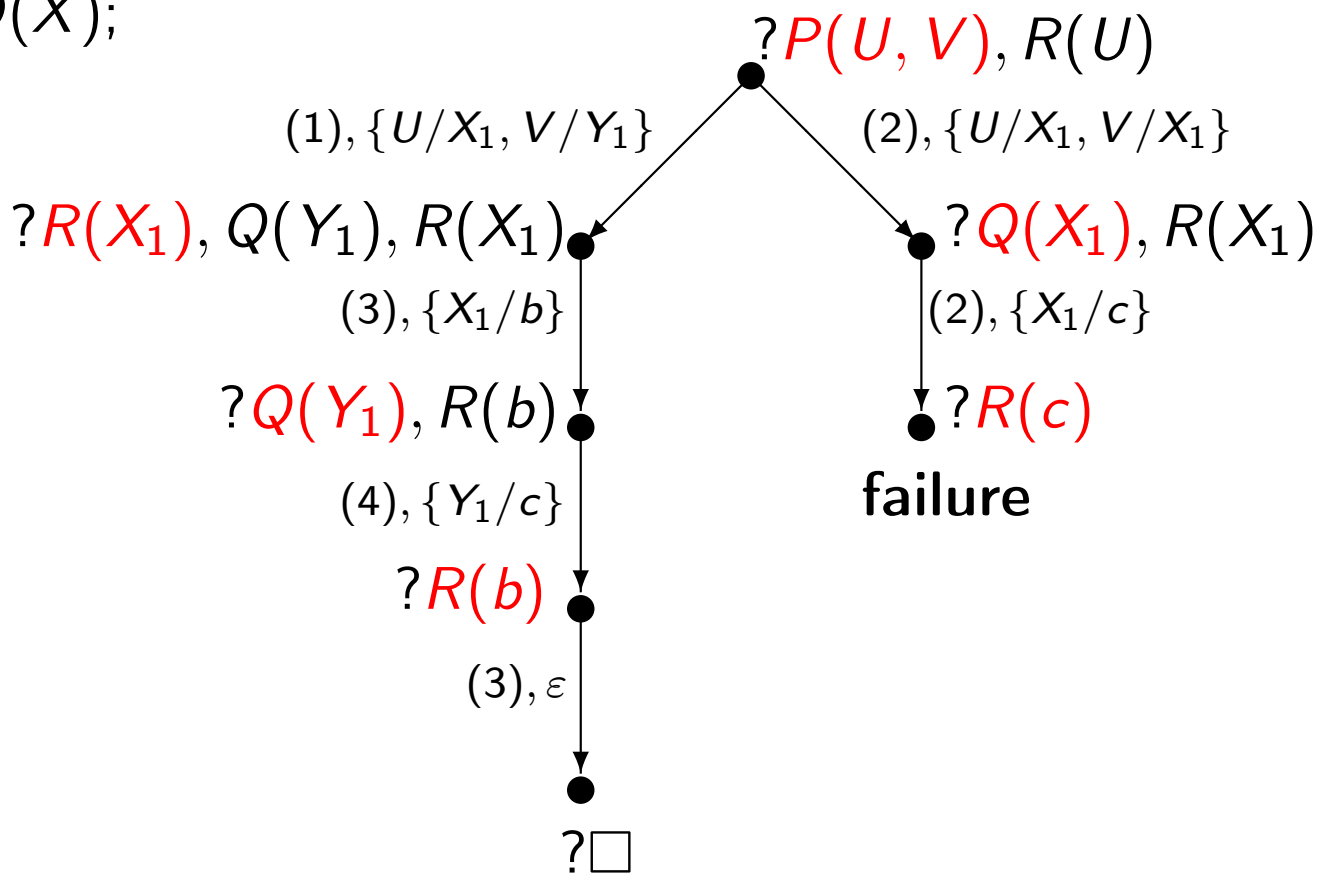
Пример обхода в ширину.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



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

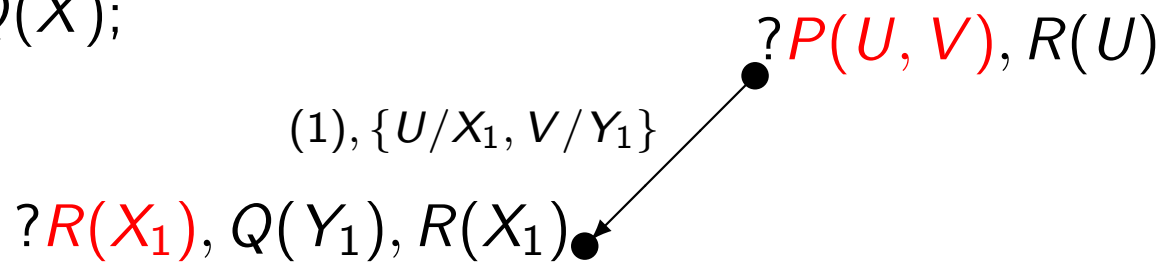
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

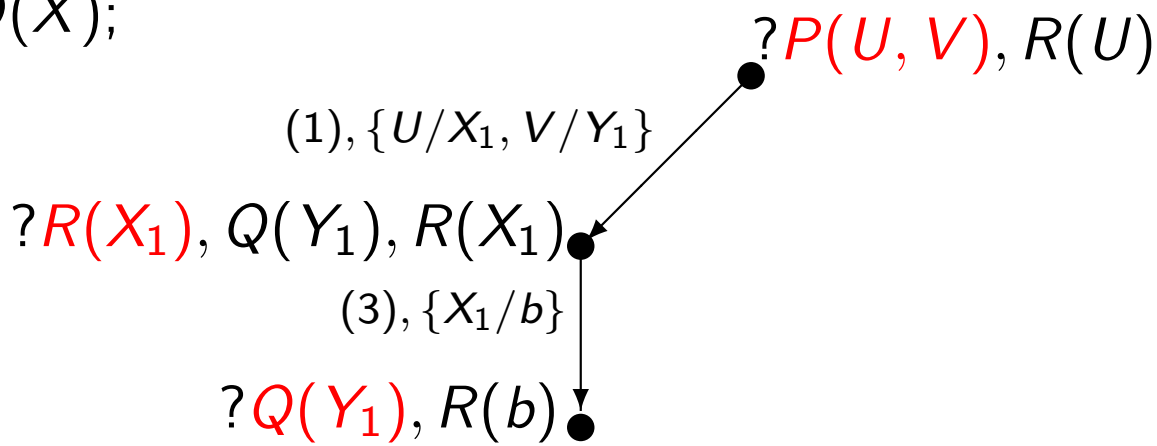
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

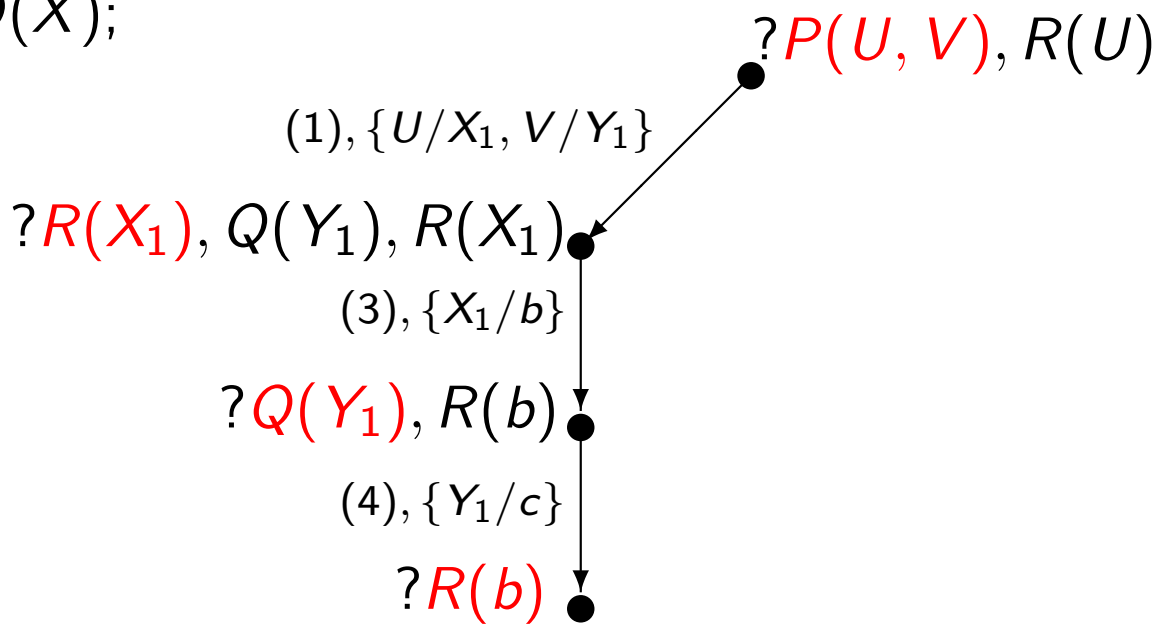
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

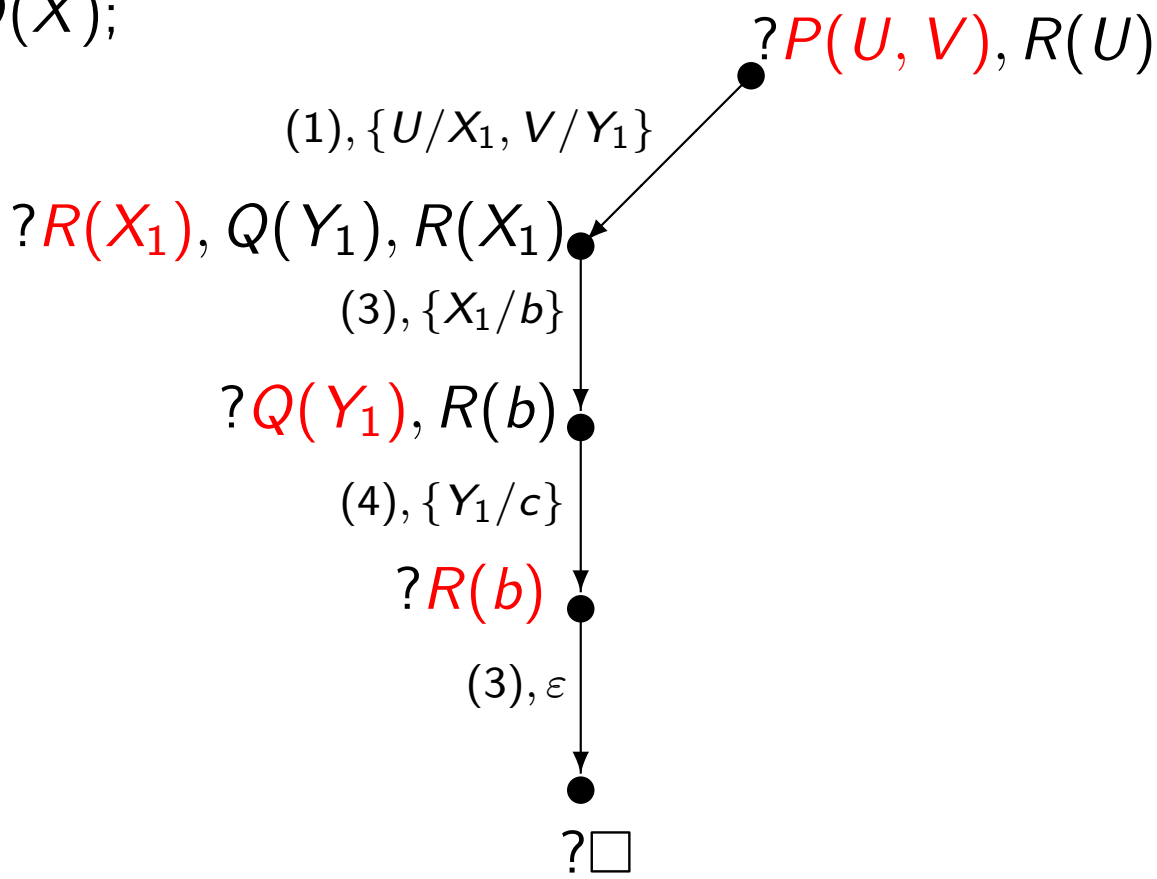
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

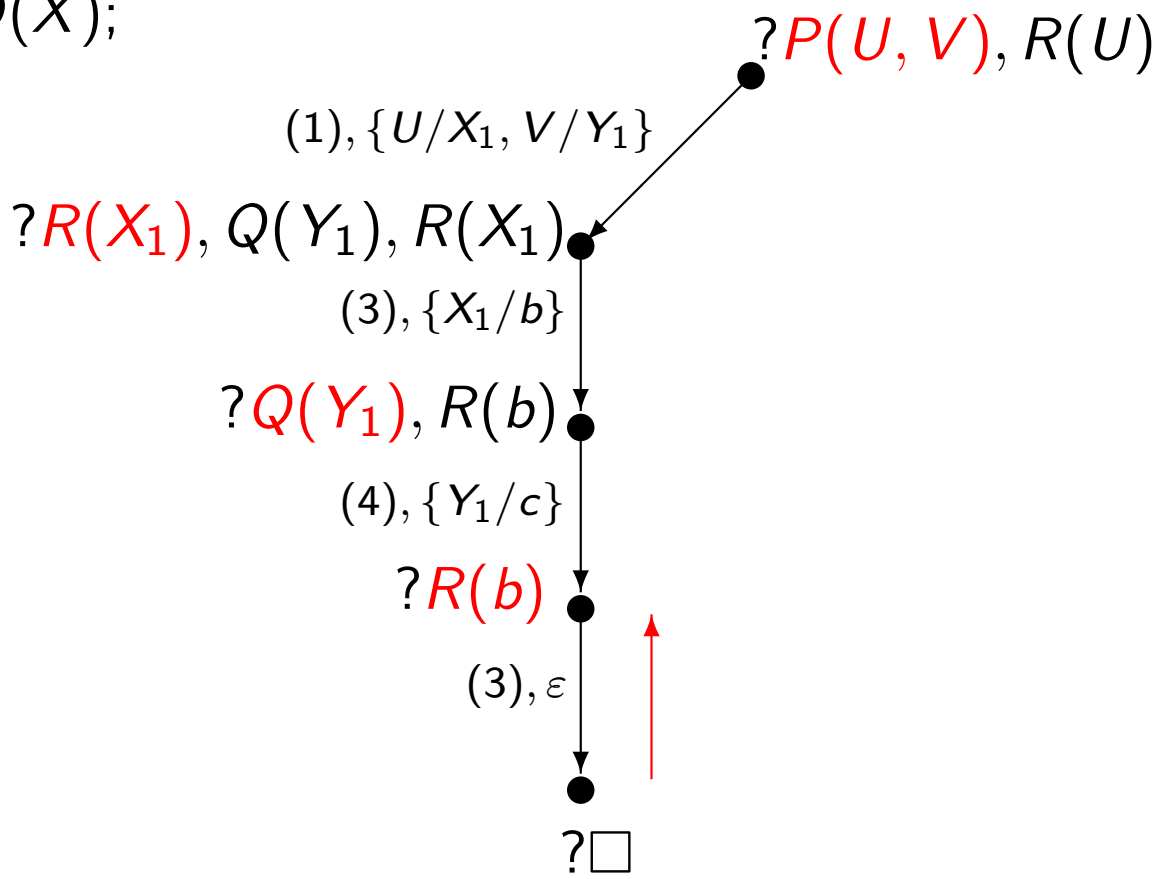
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

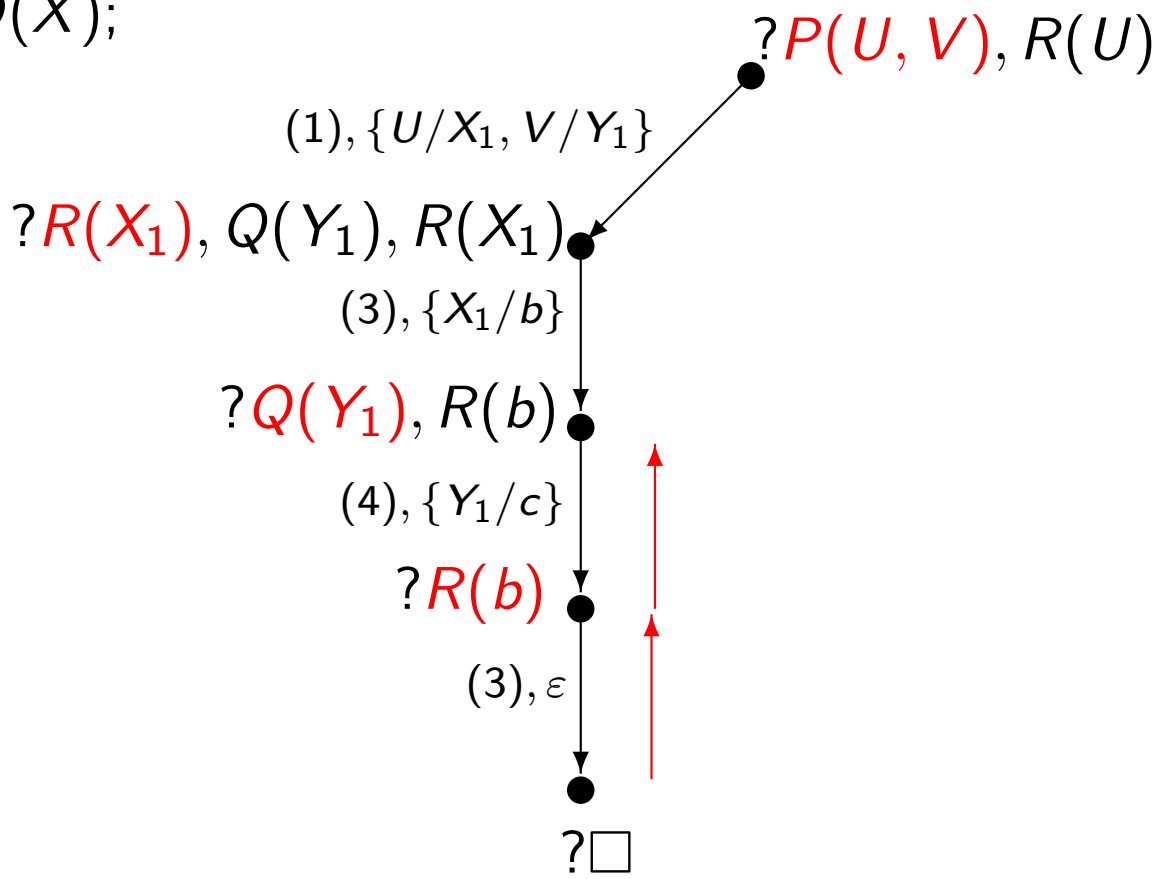
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

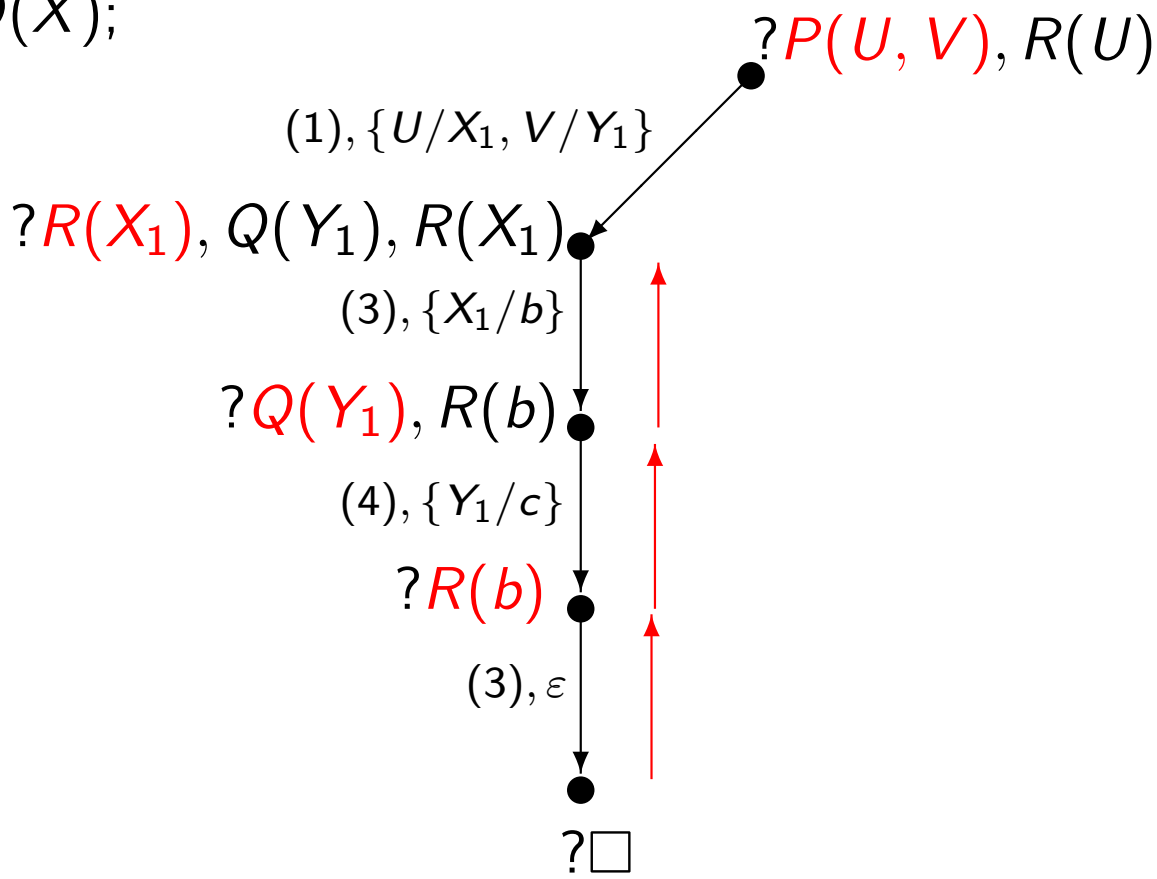
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

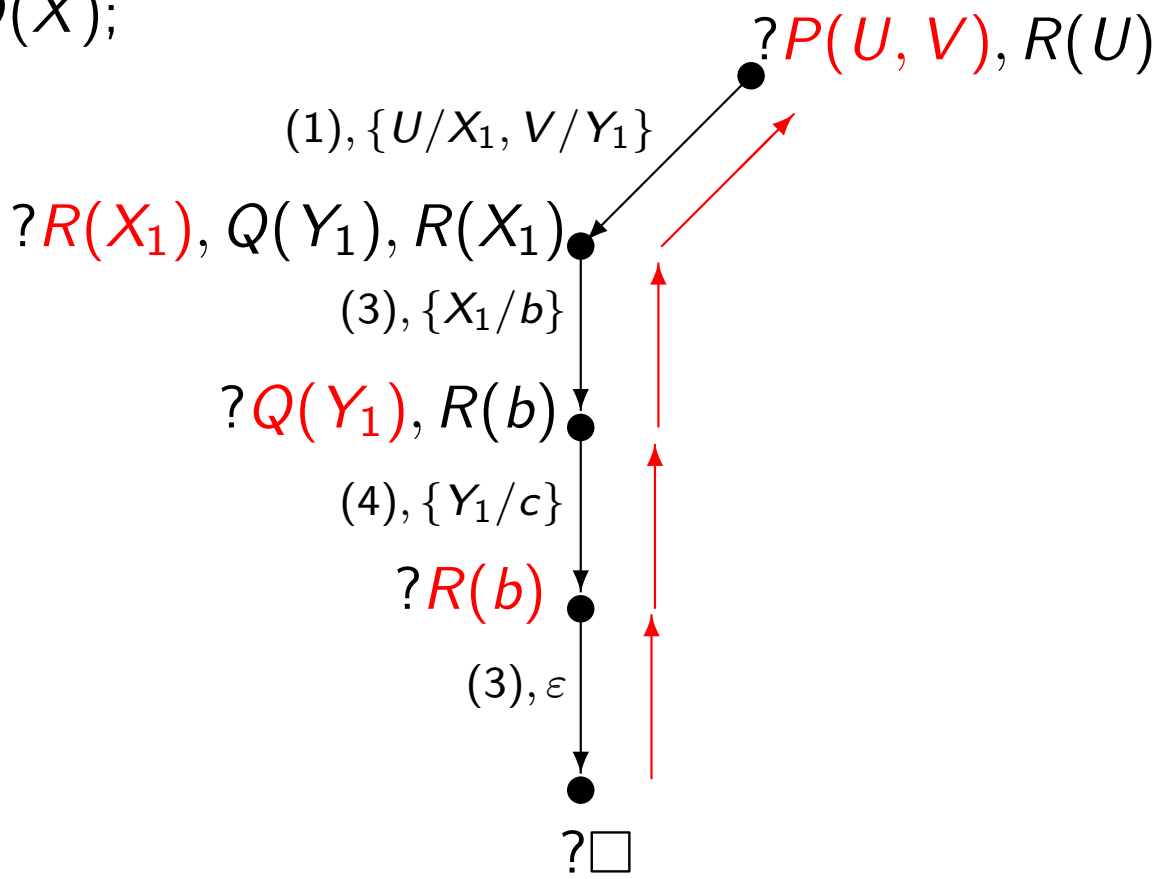
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

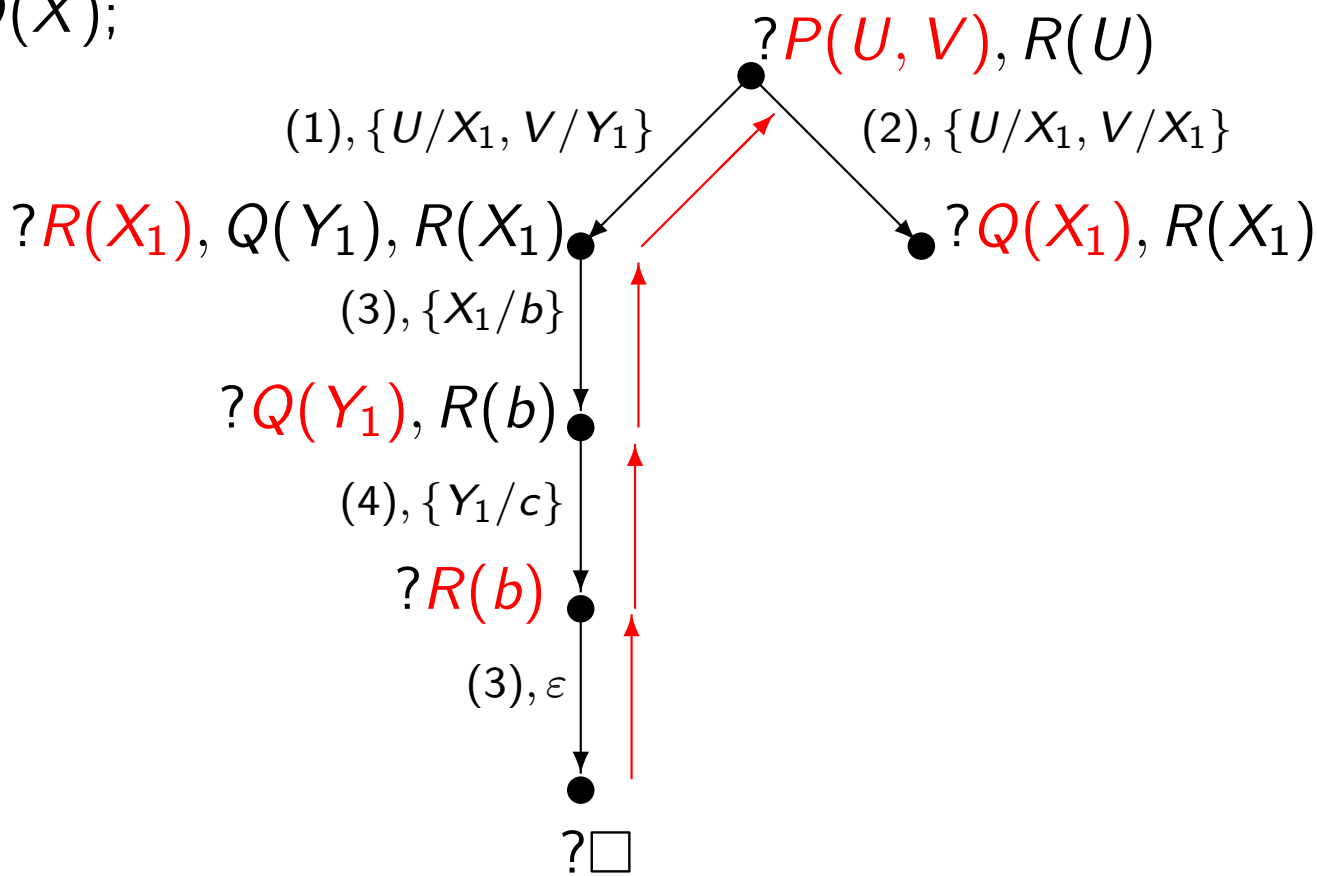
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

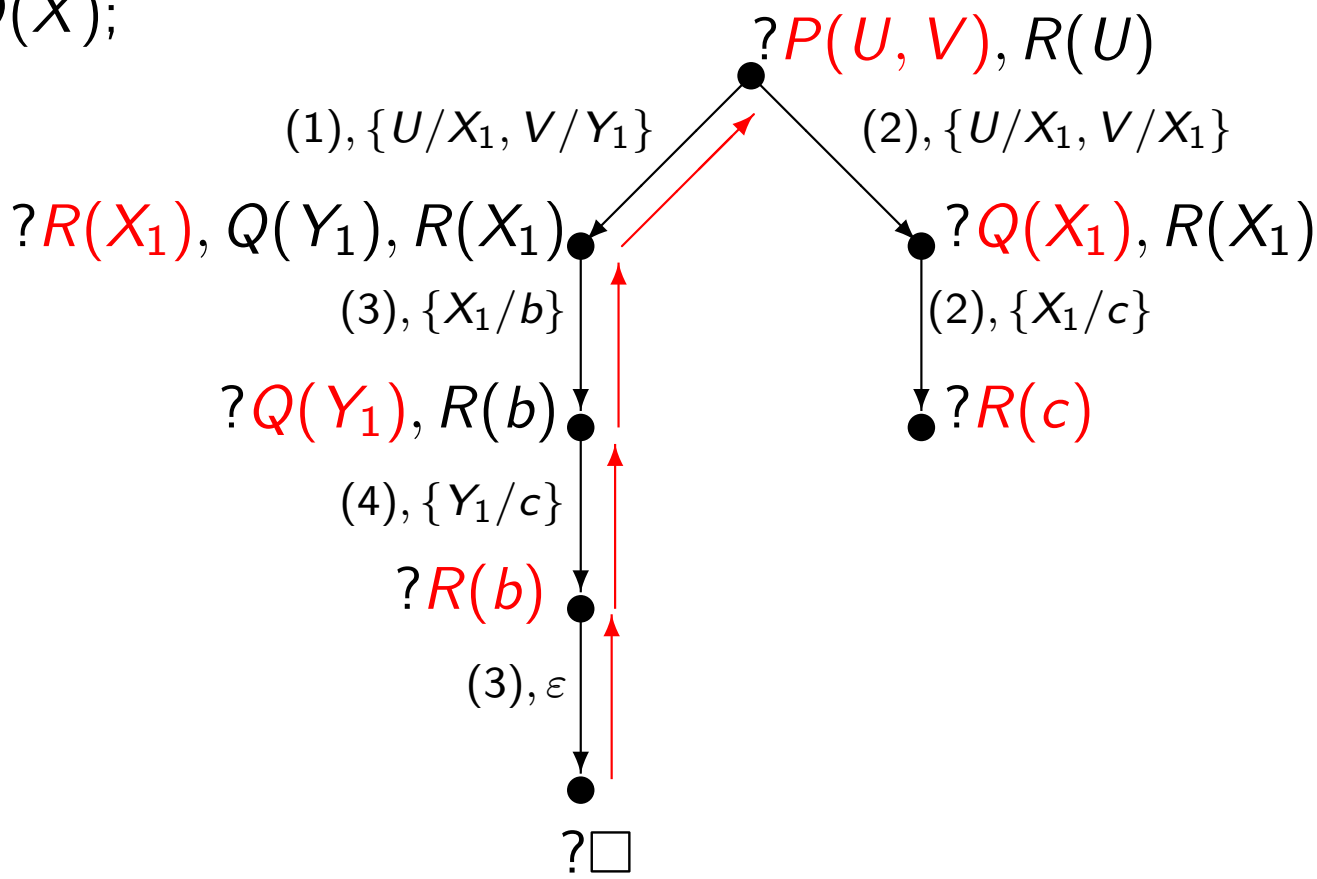
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

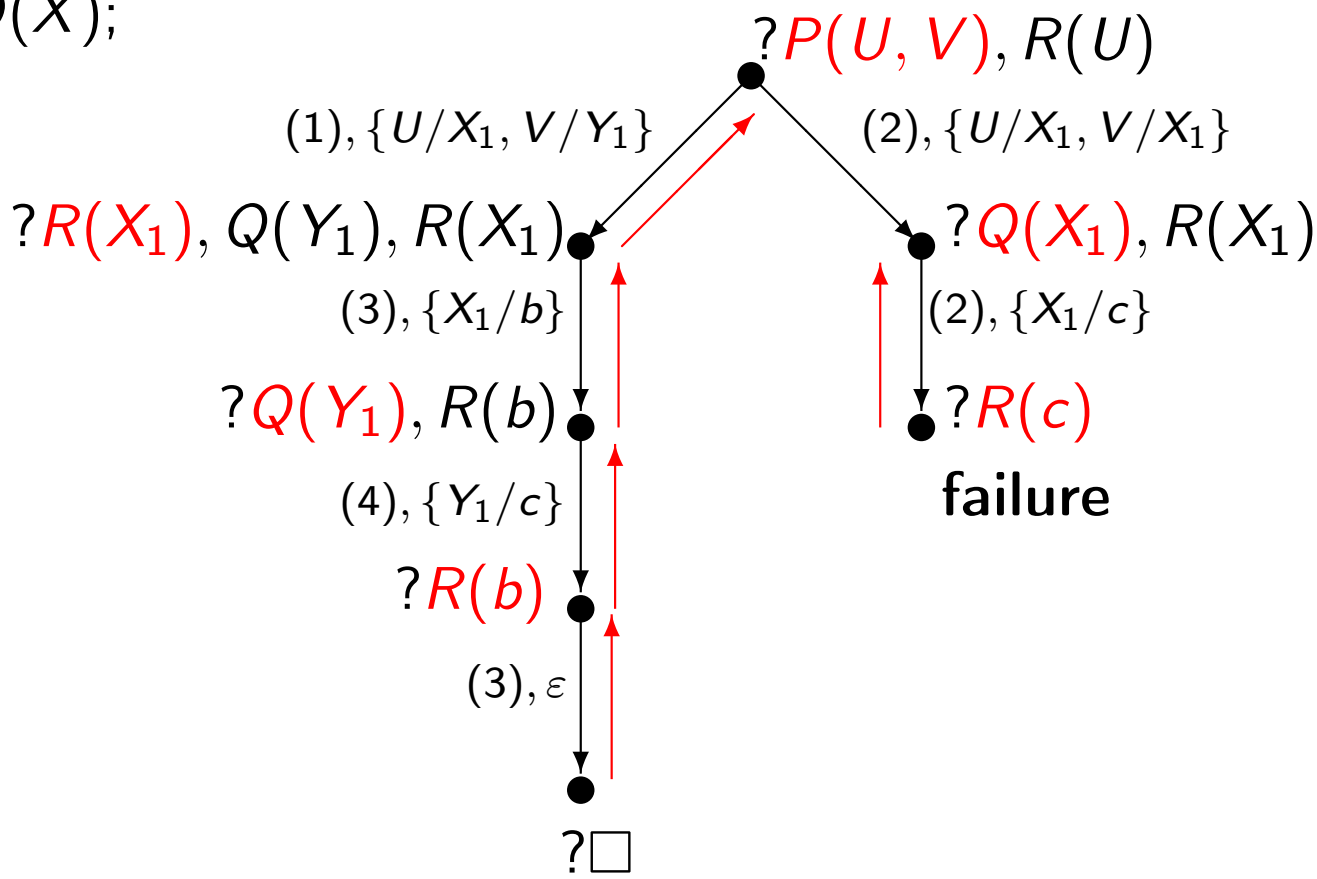
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



СТРАТЕГИИ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

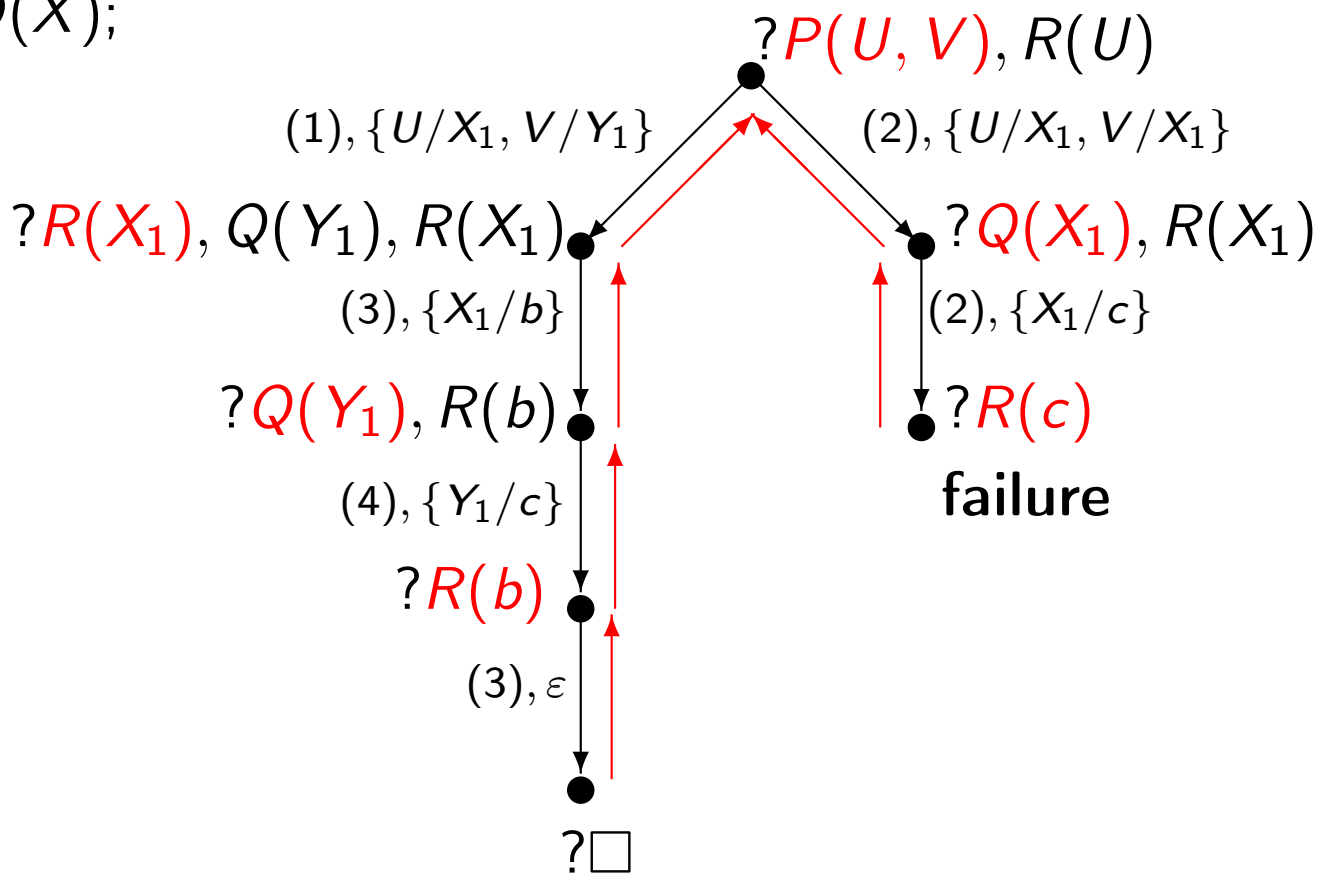
Пример обхода в глубину с возвратом.

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

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

$R(b) \leftarrow;$

$Q(c) \leftarrow;$



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

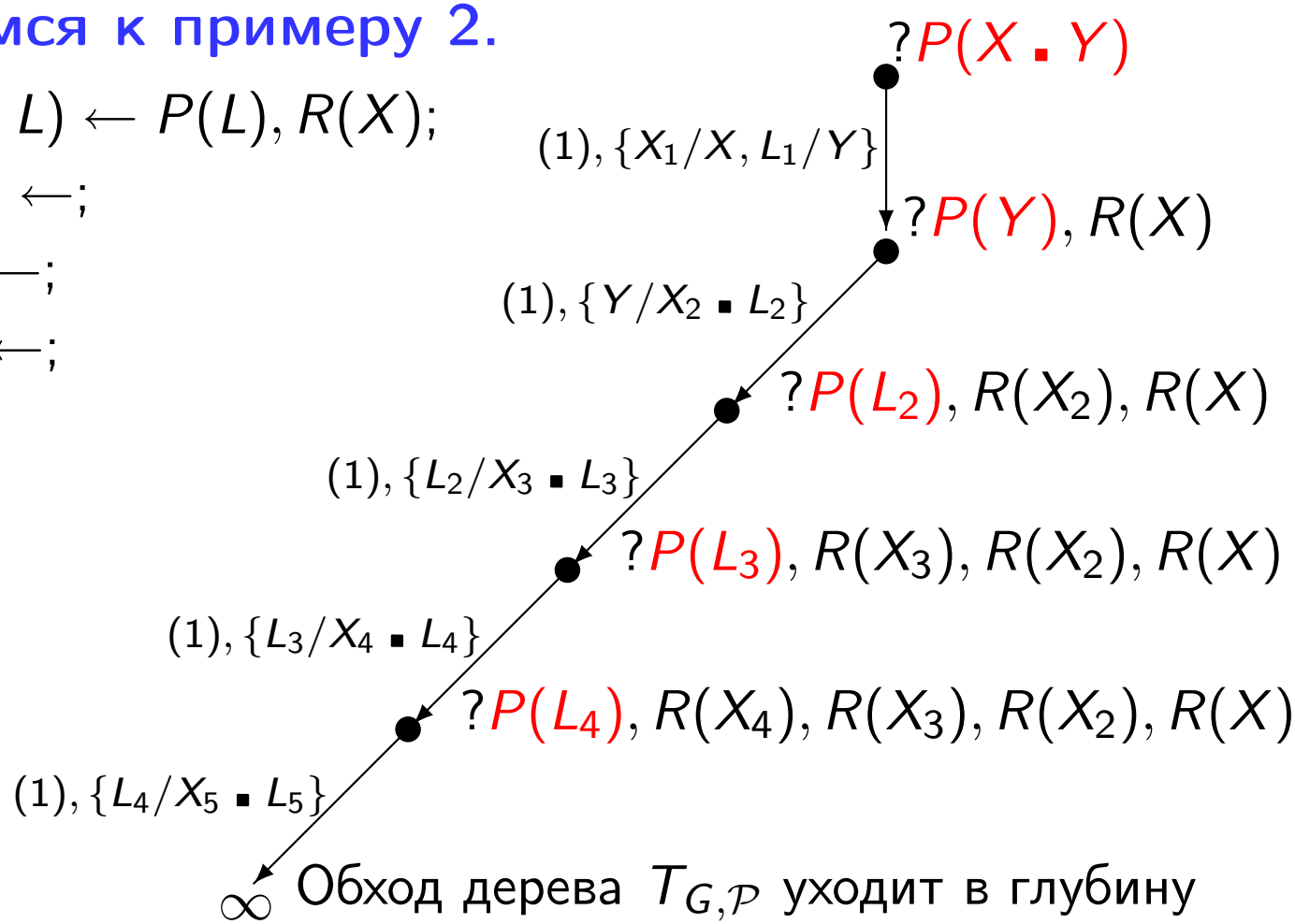
Обратимся к примеру 2.

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

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

$R(a) \leftarrow;$

$R(c) \leftarrow;$



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

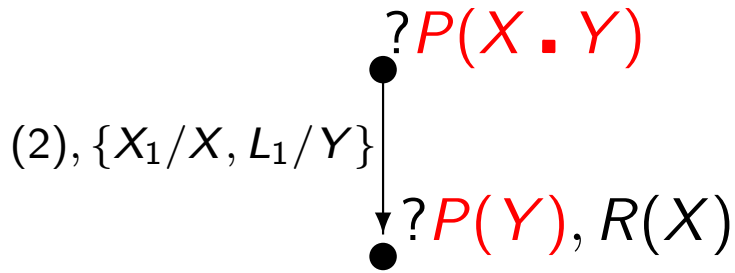
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

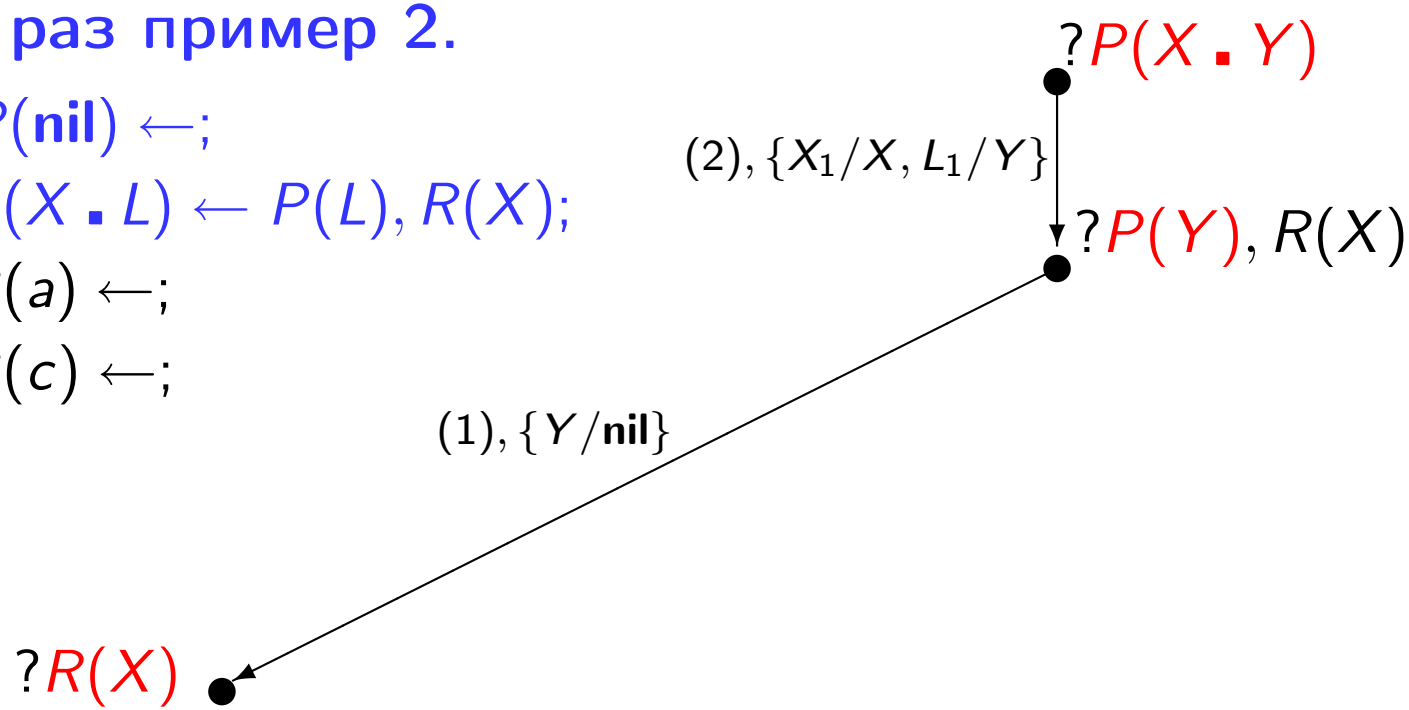
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

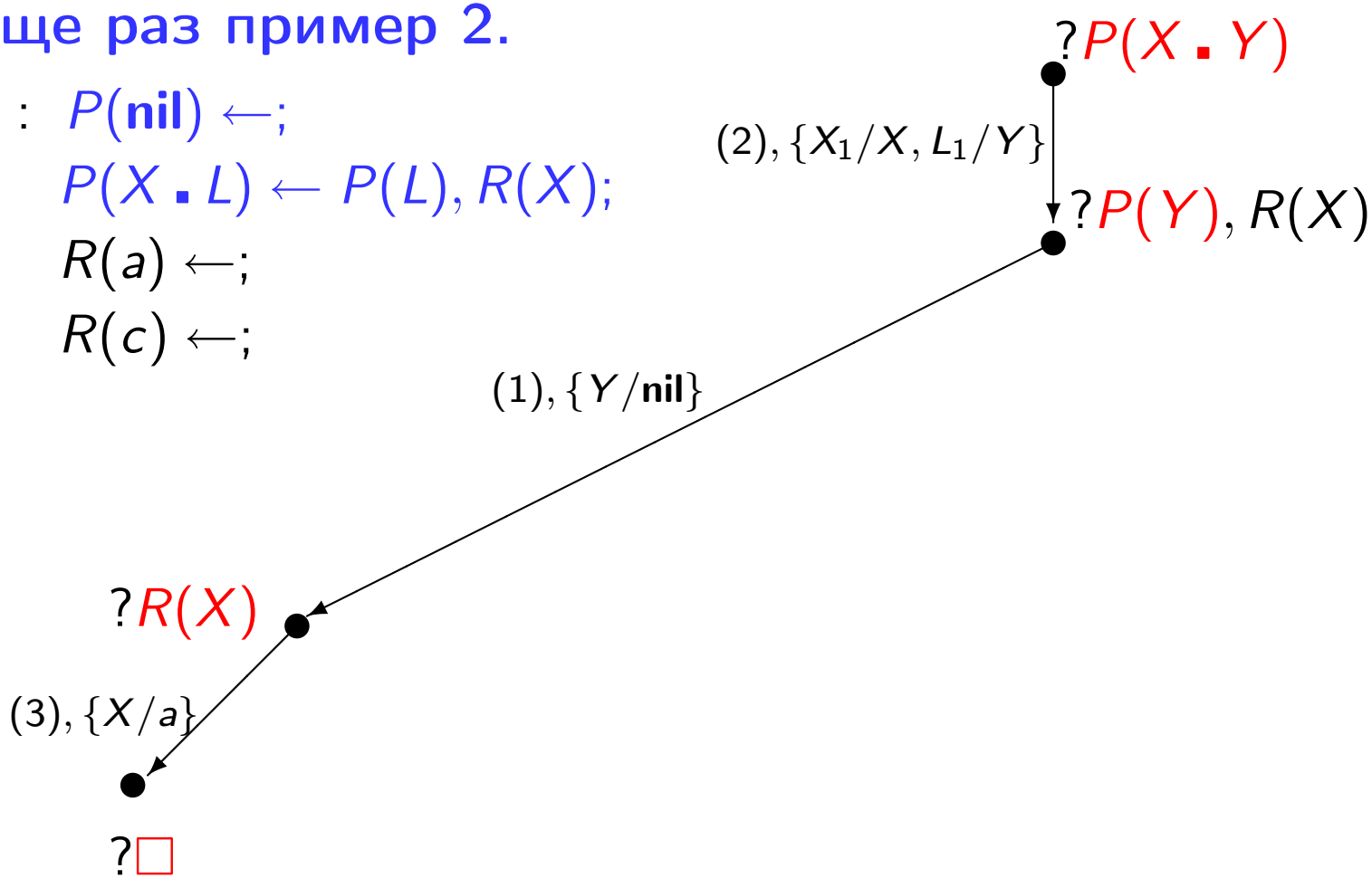
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

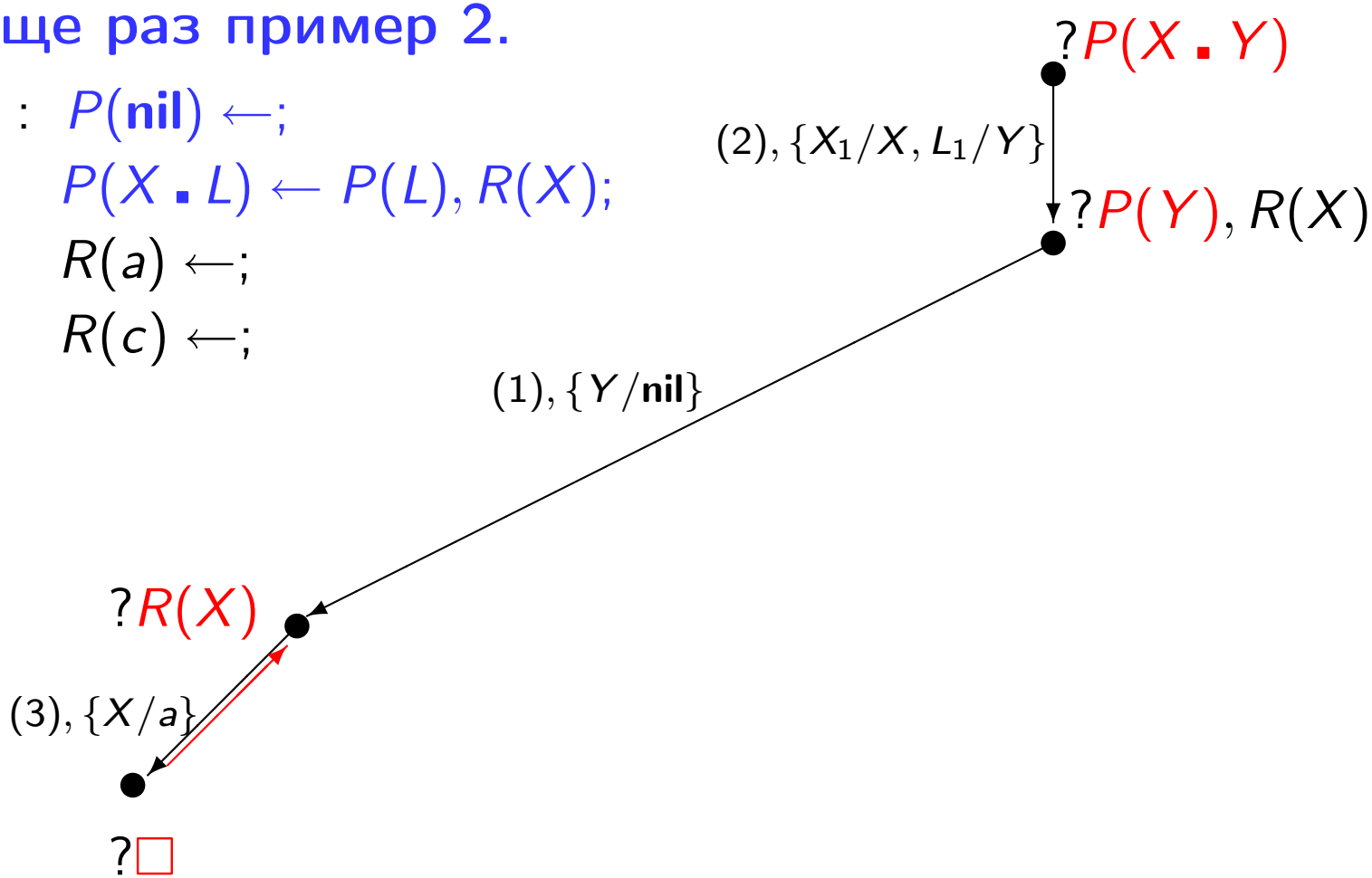
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

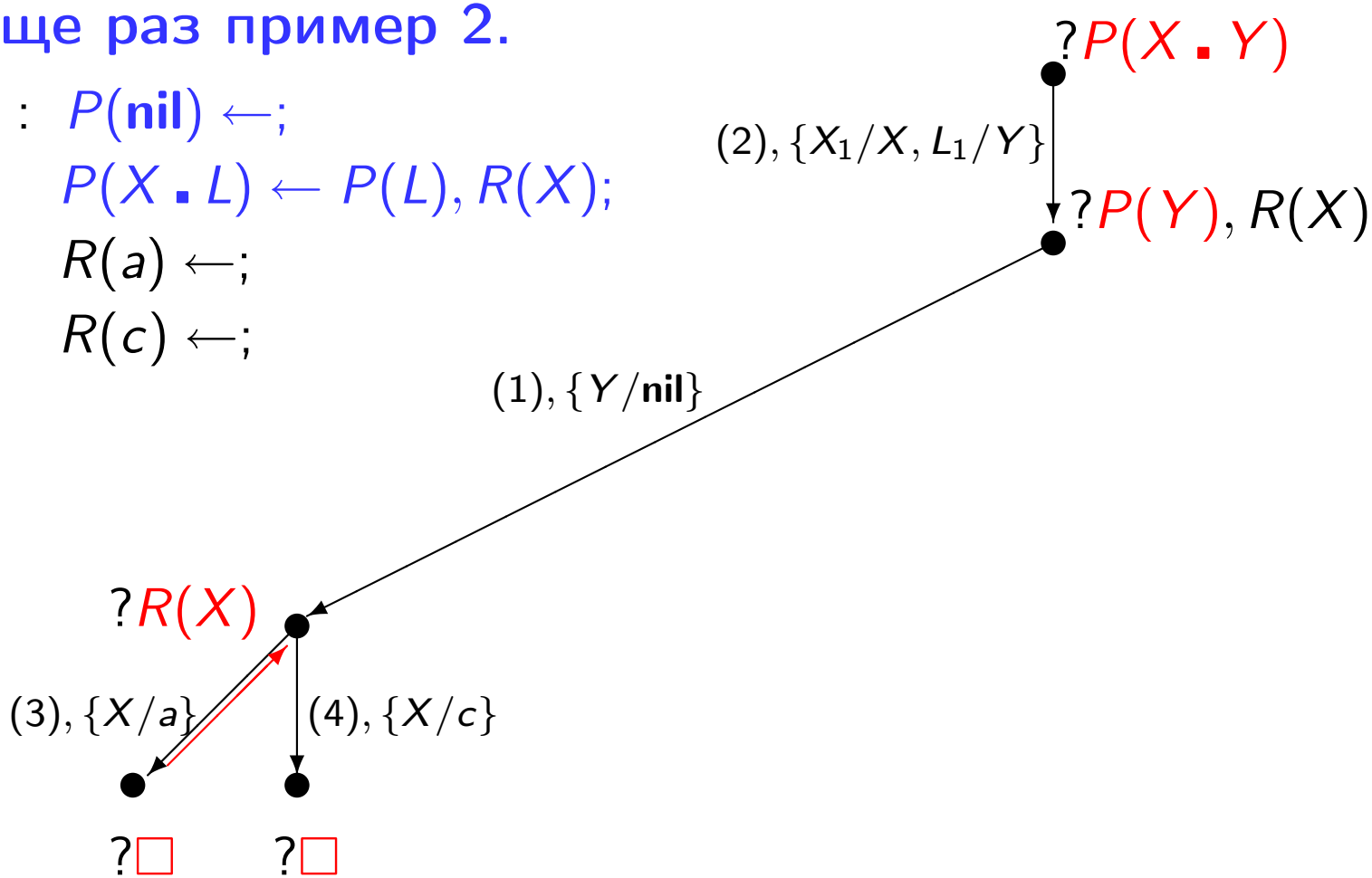
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

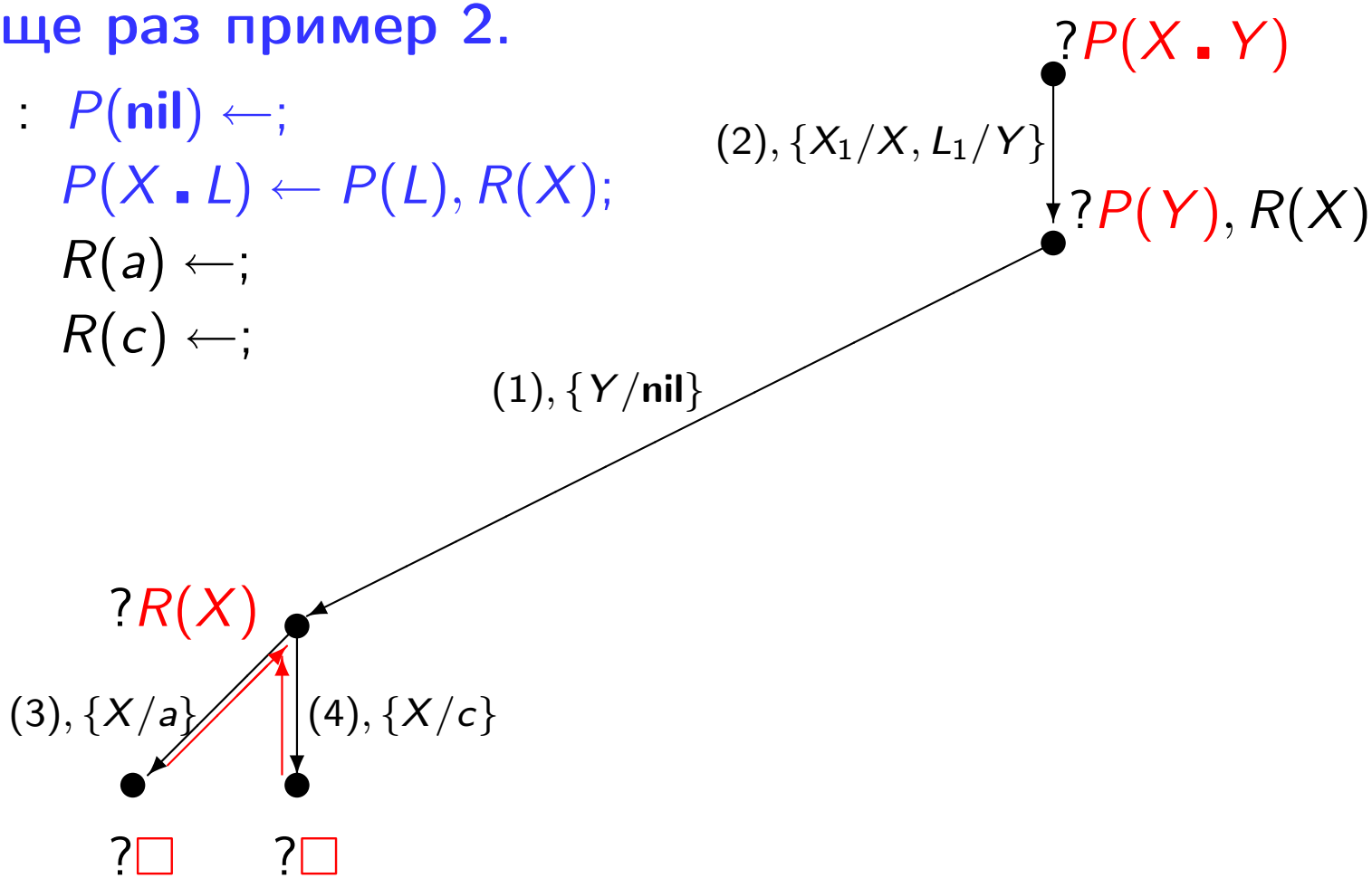
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

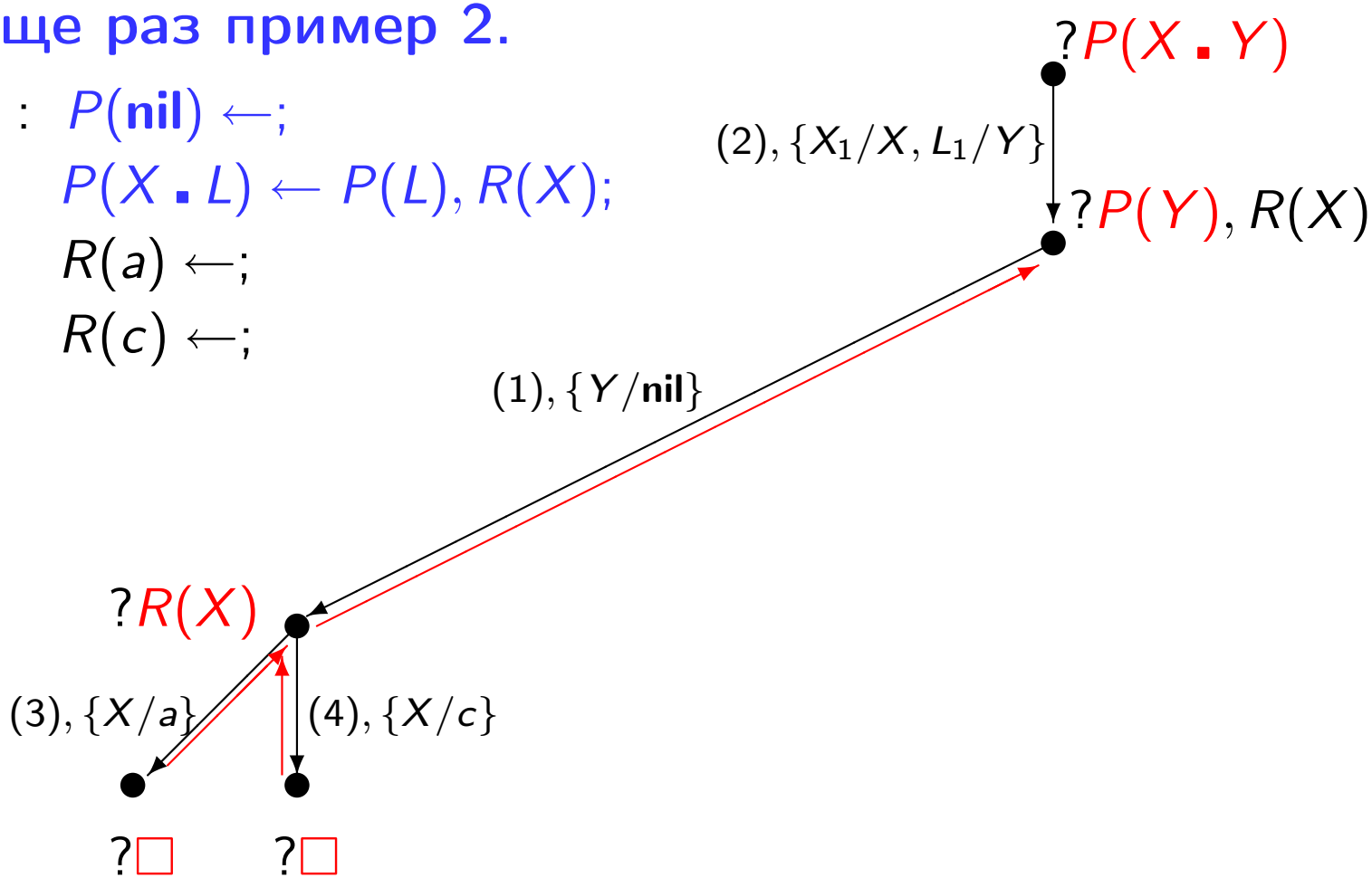
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

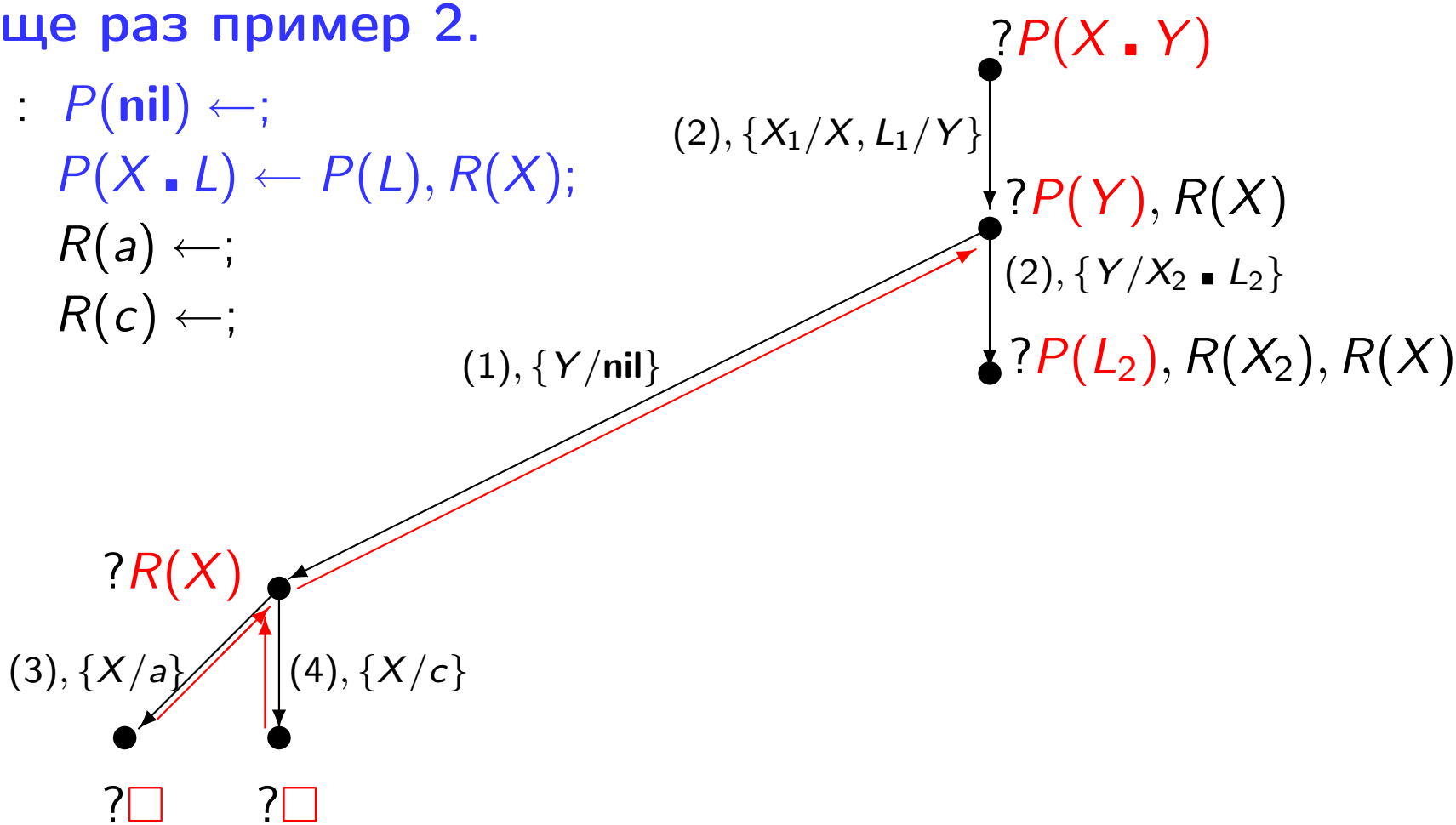
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

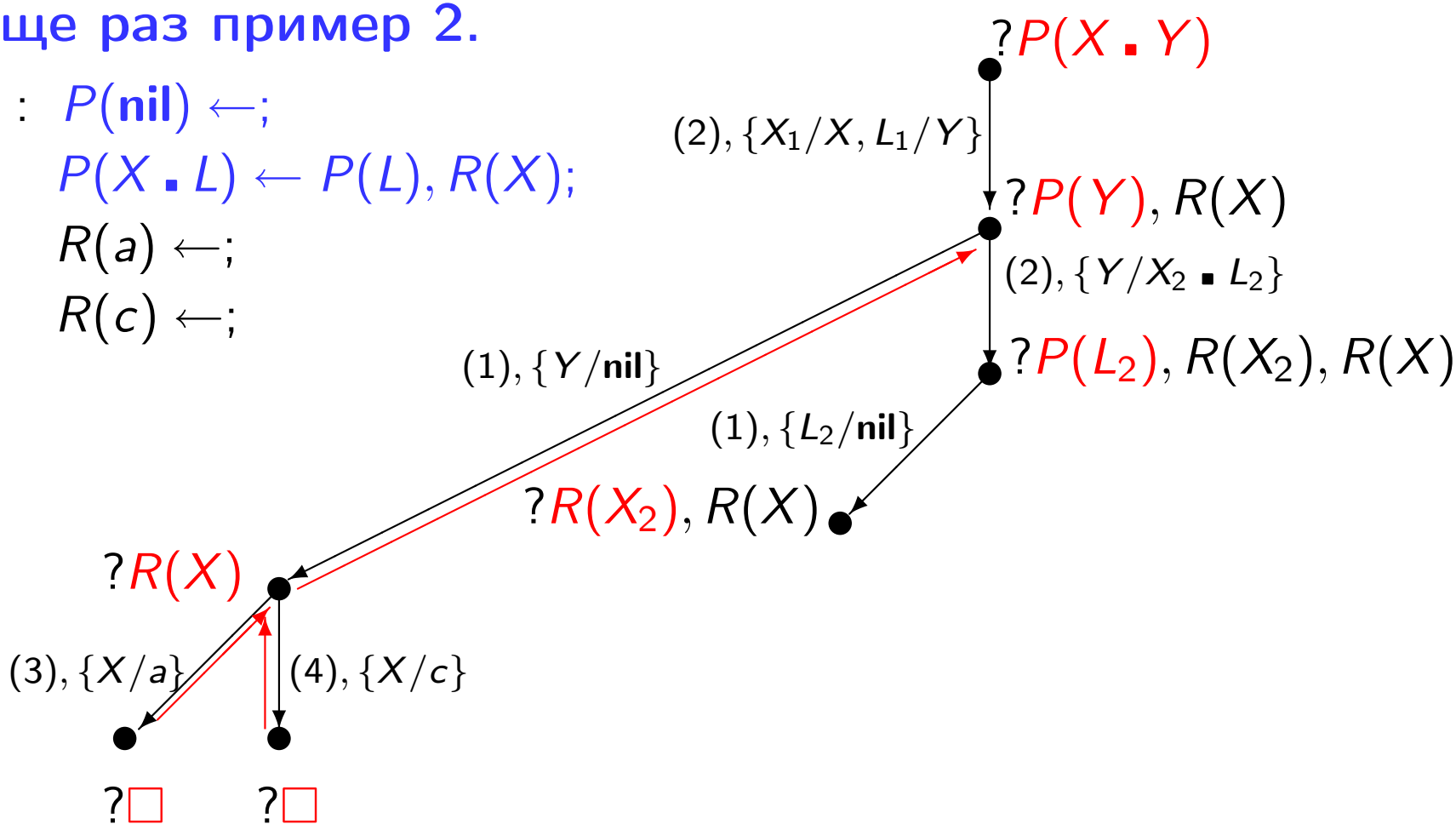
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

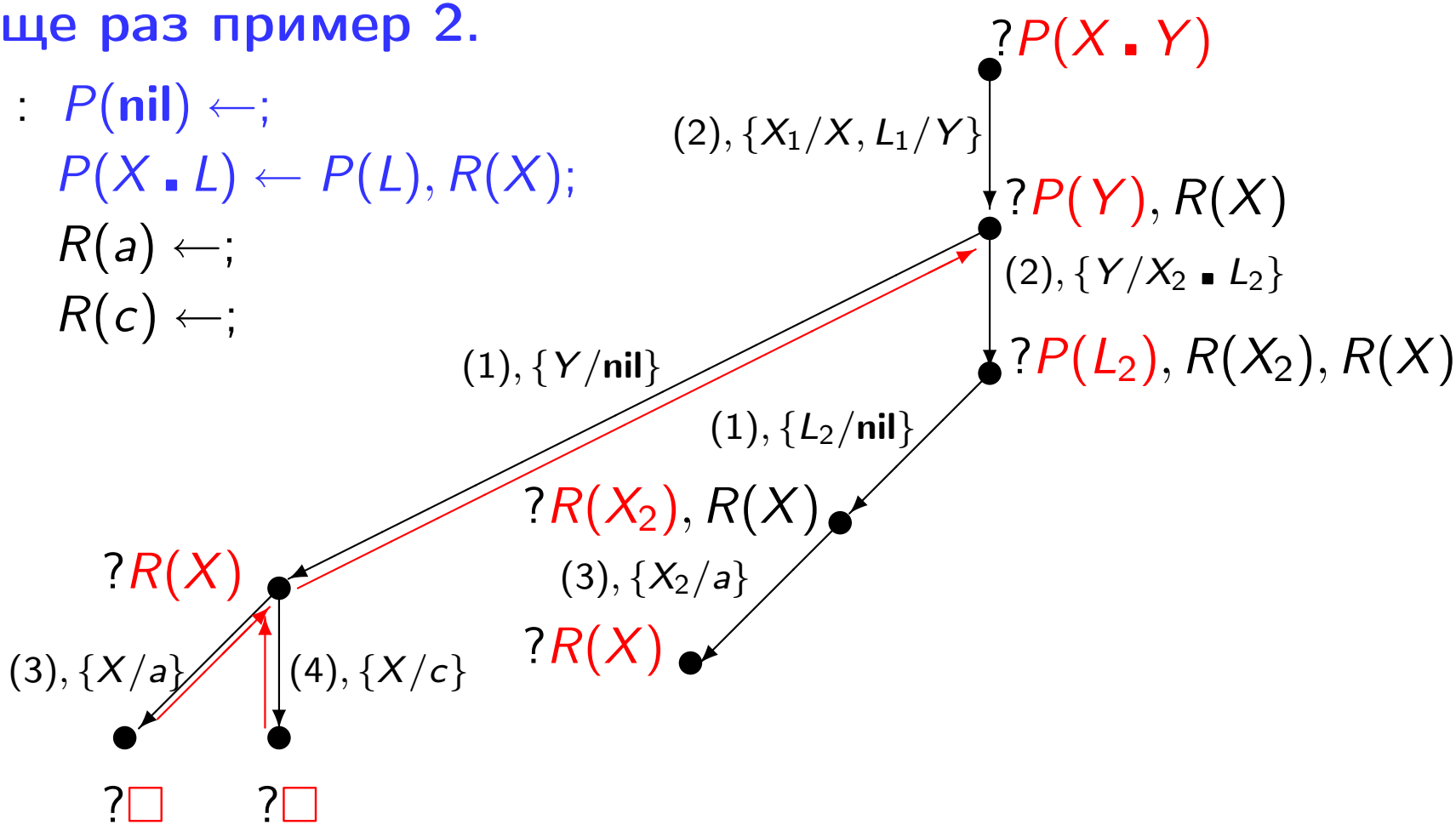
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

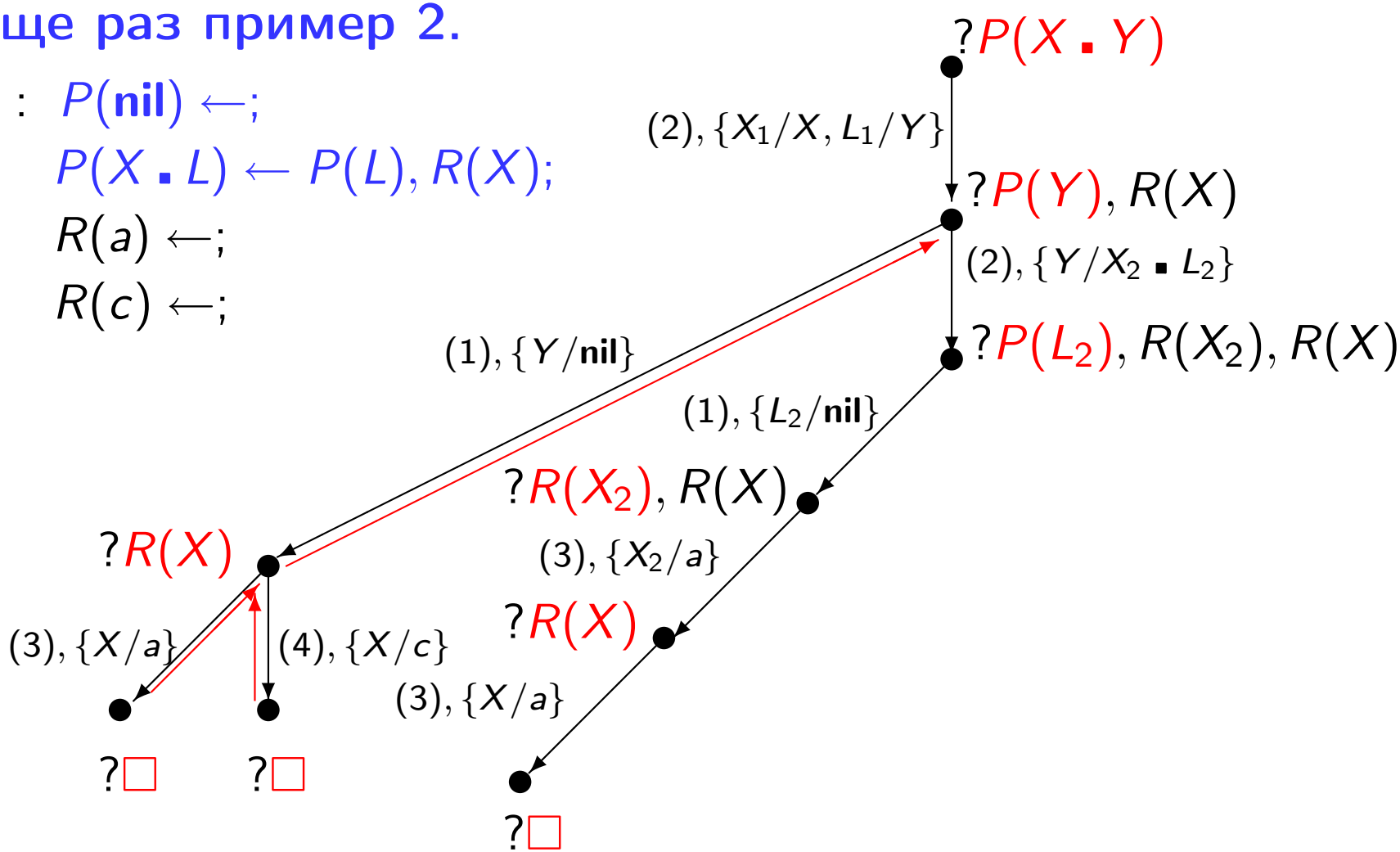
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



ДЕРЕВЬЯ ВЫЧИСЛЕНИЙ ЛОГИЧЕСКИХ ПРОГРАММ

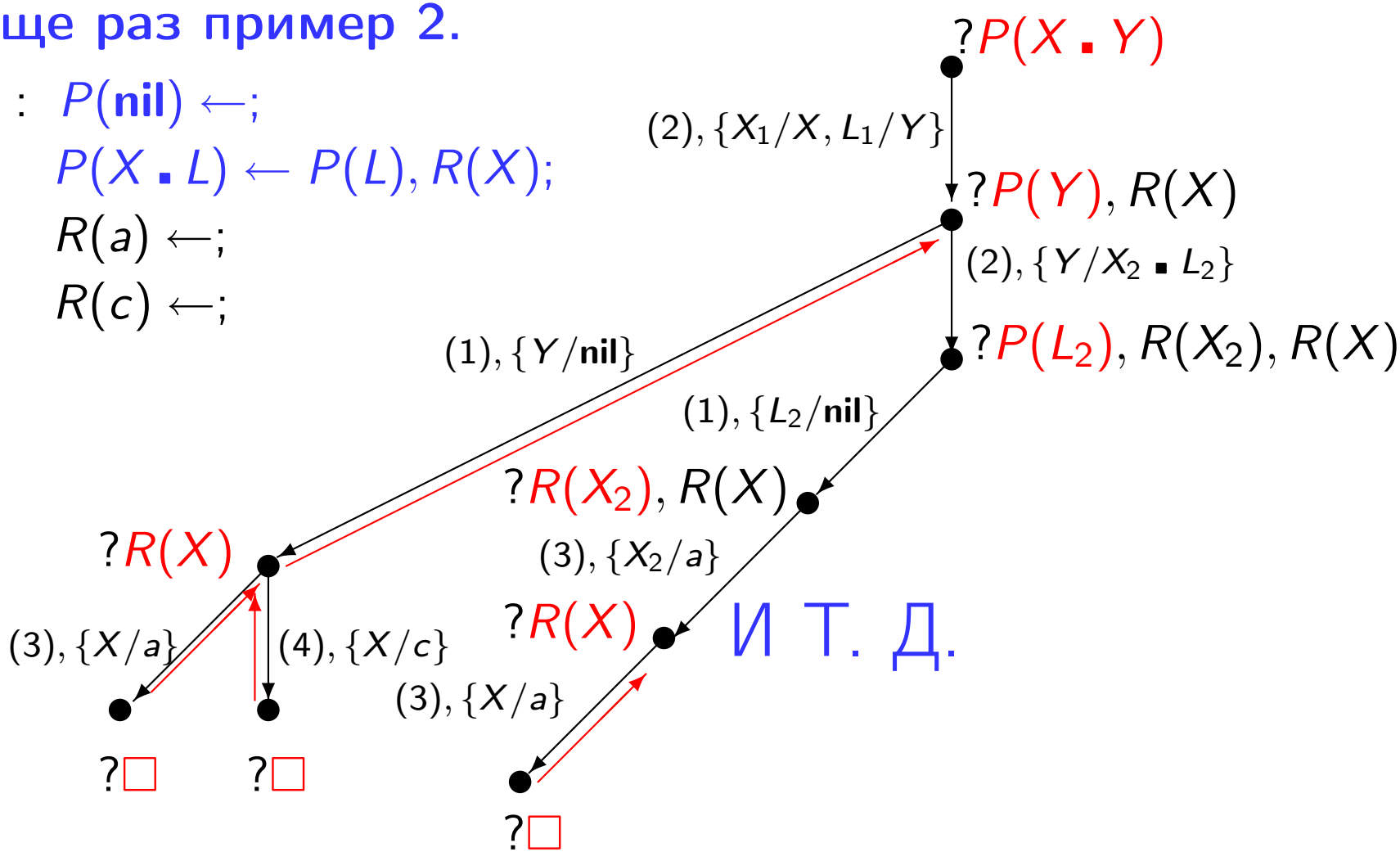
Еще раз пример 2.

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

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

$R(a) \leftarrow$;

$R(c) \leftarrow$;



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.