

Математическая логика и логическое программирование

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

Блок 46

Логические программы:
оператор отрицания,
SLDNF-резолуция

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

ВМК МГУ, 2025, сентябрь–декабрь

Оператор отрицания (not)

Оператор отрицания **not** — это особый встроенный оператор, устроенный так:

- ▶ **not** — это не предикатный символ, не функциональный символ и не константа
- ▶ На месте атома в запросе или в теле правила может быть записано выражение **not**(A),¹ где A — это атом
- ▶ В декларативной семантике **not**(A) отвечает формуле $\neg A$ в допущении замкнутости мира

А с операционной семантикой оператора **not** попробуем разобраться отдельно

Чтобы не перегружать материал тем, что относится к общему случаю и представляет в основном теоретический интерес, ограничимся семантикой **not** для **стандартной стратегии вычисления**

¹ В Prolog это записывается так: «\+ A »

Оператор отрицания (not)

Хотелось бы устроить вычисление логической программы \mathcal{P} так, чтобы для подцели **not**(A) интерпретатор программ хотя бы для основных атомов A продолжал вычисление, если $\Phi_{\mathcal{P}} \models_{cwa} \neg A$, а иначе констатировал тупик

Проверка этого соотношения для ХЛП \mathcal{P} равносильна проверке того, что не существует ни одного успешного вычисления \mathcal{P}

При обсуждении теоремы Чёрча был показано, что задача такой проверки алгоритмически неразрешима (и тем более неразрешима для ХЛП с оператором **not**)

Значит, в полной мере и эффективно воплотить допущение замкнутости мира в операционной семантике логических программ не получится

Придётся пойти на компромисс и учесть оператор **not** «достаточно разумно», сохранив эффективность вычислений

SLDNF-резолюция

Правило SLDNF-резолюции (SLD with Negation as Failure) — это аналог правила SLD-резолюции, использующийся для логических программ с отрицанием

Есть несколько строгих формулировок этого правила, и для примера обсудим формулировку, наиболее приближенную к «реальному» вычислению логических программ согласно **стандартной стратегии**

Как и для встроенных предикатов, семантику **not** можно задать как сочетание **критерия выполнимости** и **унификатора**:

- ▶ Критерий выполнимости **not**(A): строится (обходится) дерево вычислений для запроса $?A$ (**вспомогательное дерево**), и обход этого дерева **вполне неуспешен** согласно изложенному далее
- ▶ Унификатор: ϵ

Остальные понятия для вычислений программ (успешное вычисление, результат вычисления, вычислимый ответ, дерево вычислений) вводятся так же, как для «SLD», с заменой «SLD» на «SLDNF»

SLDNF-резолюция

При построении вспомогательного дерева возможны три исхода:

1. В процессе обхода дерева получен \square
 - ▶ Обнаружены успешное вычисление и вычисленный им ответ
 - ▶ В этом случае обход дерева **успешен**
2. Дерево построено, оно конечно, и в нём нет ни одного \square
 - ▶ По итогам обхода не обнаружено ни одно успешное вычисление
 - ▶ В этом случае обход дерева **вполне неуспешен**
3. Дерево и возникающие вспомогательные деревья обходятся бесконечно, и в строящемся фрагменте дерева нет ни одного \square
 - ▶ Значит, программа «зациклилась» при проверке вполне-неуспешности дерева
 - ▶ В этом случае обход дерева **неуспешен, но не вполне**
 - ▶ Соответствующее вычисление программы (для исходного запроса) признаётся бесконечным («сингулярная бесконечность»), и обход исходного дерева и всех остальных вспомогательных поддеревьев немедленно завершается

SLDNF-резолюция

Теорема (о корректности SLDNF-резолюции). Для любой ХЛП с операторами отрицания \mathcal{P} и любого запроса с операторами отрицания \mathcal{Q} верно следующее: результат любого успешного SLDNF-резолютивного вычисления \mathcal{P} на \mathcal{Q} является правильным ответом в допущении замкнутости мира

Доказательство опустим: времени потратим много, а пользы будет мало

А аналогичная теорема о полноте оказывается неверной из-за

- ▶ использования стандартной стратегии вычисления (*этого можно было бы избежать*) и
- ▶ возможного появления бесконечных вспомогательных деревьев (*а этого избежать в общем случае не выйдет*)

SLDNF-резолюция

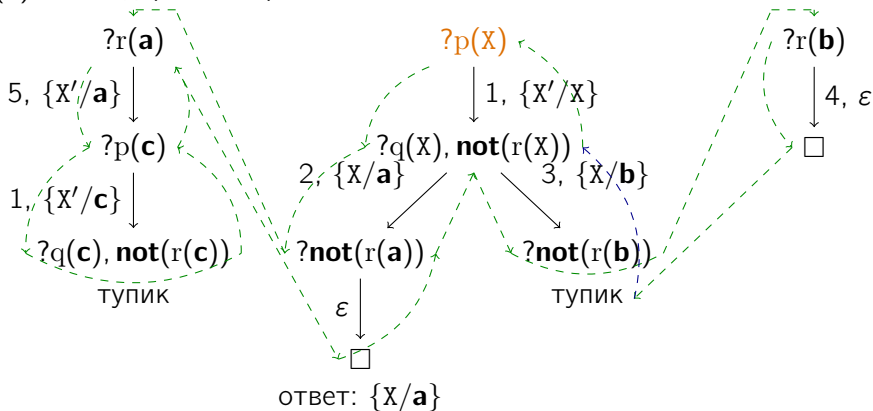
Примеры

1 : $p(X) \leftarrow q(X), \text{not}(r(X))$;

2 : $q(a)$; 3 : $q(b)$;

4 : $r(b)$; 5 : $r(X) \leftarrow p(c)$;

Дерево SLDNF-резолютивных вычислений этой программы для запроса $?p(X)$ и стандартной стратегии вычисления:



SLDNF-резолюция

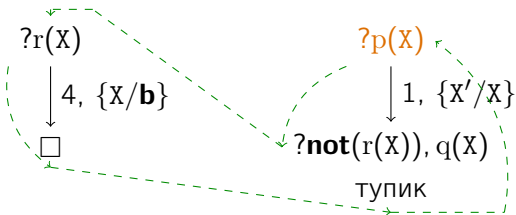
Примеры

1 : $p(X) \leftarrow \text{not}(r(X)), q(X);$

2 : $q(\mathbf{a});$ 3 : $q(\mathbf{b});$

4 : $r(\mathbf{b});$ 5 : $r(X) \leftarrow p(\mathbf{c});$

Дерево SLDNF-резолутивных вычислений этой программы для запроса $?p(X)$ и стандартной стратегии вычисления:



SLDNF-резолюция

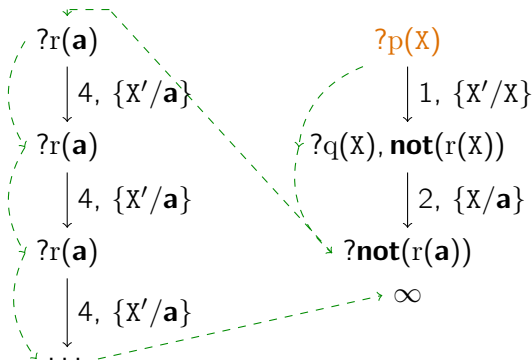
Примеры

1 : $p(X) \leftarrow q(X), \text{not}(r(X));$

2 : $q(\mathbf{a});$ 3 : $q(\mathbf{b});$

4 : $r(X) \leftarrow r(X);$

Дерево SLDNF-резолютивных вычислений этой программы для запроса $?p(X)$ и стандартной стратегии вычисления:

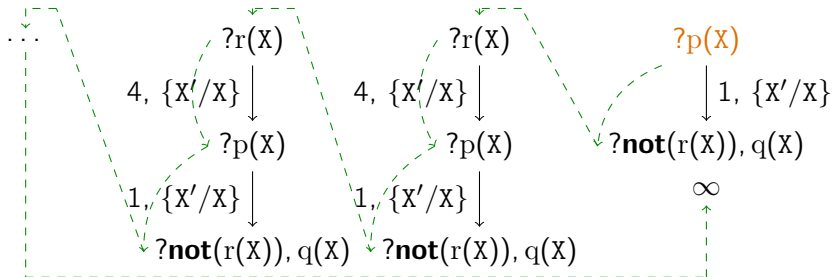


SLDNF-резолюция

Примеры

$$1 : p(X) \leftarrow \mathbf{not}(r(X)), q(X);$$
$$2 : q(\mathbf{a}); \quad 3 : q(\mathbf{b});$$
$$4 : \quad r(X) \leftarrow p(X);$$

Дерево SLDNF-резолутивных вычислений этой программы для запроса $?p(X)$ и стандартной стратегии вычисления:



SLDNF-резолюция

Более «осмысленный» пример

Логическая программа, такая что $p(X, L, M)$ вычисляет в X произвольный элемент списка L , не содержащийся в списке M :

```
1 : p(X, L, M) ← e(X, L), not(e(X, M));  
2 : e(X, X.L);  
3 : e(X, Y.L) ← e(X, L);
```

Единственный правильный ответ на запрос $?p(X, \mathbf{a.b.nil}, \mathbf{b.c.nil})$ в допущении замкнутости мира: $\{X/\mathbf{a}\}$

SLDNF-резолюция

Более «осмысленный» пример

1 : $p(X, L, M) \leftarrow e(X, L), \text{not}(e(X, M));$

2 : $e(X, X.L);$ 3 : $e(X, Y.L) \leftarrow e(X, L);$

Дерево SLDNF-резольтивных вычислений этой программы для $?p(X, \mathbf{a.b.nil}, \mathbf{b.c.nil})$ и стандартной стратегии вычисления:

