Ефремов Евгений Леонидович

МАТЕМАТИЧЕСКАЯ ЛОГИКА

Конспект лекций

Владивосток

2021 г.

ОГЛАВЛЕНИЕ

ПРИЛОЖЕНИЕ А

ЯЗЫК ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ ПРОЛОГ

Метод резолюций в логике предикатов лежит в основе декларативного языка логического программирования ПРОЛОГ (PROLOG, франц. PROgrammation en LOGique). ПРОЛОГ появился в 1973 году, когда группа исследователей из Марсельского университета под руководством Алана Колмероэ создала программу для доказательства теорем, которая была реализована на языке Фортран. Впоследствии этот продукт и получил название ПРОЛОГ. Первые годы ПРОЛОГ не находил широкого практического применения. Однако в 1981 г. было объявлено о японском проекте создания ЭВМ пятого поколения, в основе программного обеспечения которых предполагалось использовать логическое программирование. Этот проект создания ЭВМ для обработки знаний вызвал большой резонанс во всём мире. Появились коммерческие реализации ПРОЛОГа, такие как CProlog, Quintus Prolog, Silogic Knowledge Workbench, Turbo Prolog и др. Наибольшую популярность в нашей стране получила система программирования Turbo Prolog — один из продуктов известной фирмы Borland.

Не будем ставить цель изучить все возможности этого языка. Дадим лишь его краткий обзор.

Программа на языке ПРОЛОГ включает следующие основные разделы:

— описание имён и структур объектов (*domains*);

— описание фактов — структура утверждений базы знаний (*class facts*);

— описание предикатов — структура отношений, существующих между объектами (*class predicates*);

— раздел целевых утверждений (*goal*); этот раздел может отсутствовать, в этом случае программа будет запрашивать целевое утверждение при запуске;

— описание фактов и правил, описывающих отношения (*clauses*).

Имена объектов (констант) в ПРОЛОГе пишутся с маленькой буквы, а переменных — с большой.

Рассмотрим, как можно описать на ПРОЛОГе задачу из некоторой предметной области, например географии.

```
implement main
    open core, console

domains
    gorod = symbol.
    strana = symbol.

class predicates
    situ : (gorod, strana) nondeterm anyflow.
clauses
    situ("Лондон", "Англия").
    situ("Киев", "Украина").
    situ("Пекин", "Китай").
    situ("Варшава", "Польша").
    situ(X, "Европа") :-
        situ(X, "Польша").
    situ(X, "Азия") :-
        situ(X, "Китай").

    run() :-
        foreach situ(X, Y) do
            write(X),
            write(" - "),
            write(Y),
            nl
        end foreach,
        _ = readLine().

end implement main

goal
    mainExe::run(main::run).
```

Выражение `situ("Лондон", "Англия")` описывает тот факт, что город Лондон находится в Англии. Ранее в программе был введён соответствующий предикат, работающий с объектами символьного типа. Вместо блока определений

```
domains
    gorod = symbol.
    strana = symbol.

class predicates
    situ : (gorod, strana) nondeterm anyflow.
```

можно было просто описать предикат

```
class predicates
    situ : (symbol, symbol) nondeterm anyflow.
```

Первый вариант предпочтительнее, поскольку позволяет оценивать семантический смысл аргументов предиката на этапе создания программы и при её отладке.

В середине раздела `clauses` описаны два *правила*. Правило задаёт новый предикат через предикаты, определённые ранее. Правило состоит из *головы* (предиката) и *тела* (последовательности предикатов). Голова отделяется от тела значком `:-`, который можно интерпретировать как слово «если». Таким образом, заключение является головой правила, а тело правила состоит из набора посылок.

Использование правил является основным способом представления знаний в интеллектуальных системах. Смысл правила состоит в том, что цель, являющаяся головой, будет истинной, если интерпретатор ПРОЛОГа сможет показать, что все выражения (подцели) в теле правила являются истинными. В правилах буква `X` (или любая другая заглавная буква, или любое слово, начинающееся с заглавной буквы) обозначает переменную, которая может принимать разные значения. Так, правило

```
    situ(X, "Европа") :-
        situ(X, "Польша").
```

означает, что любой польский город является одновременно европейским городом. Добавлением новых правил можно пополнять и модифицировать описание задачи. Если мы хотим описать тот факт, что все города Франции являются одновременно европейскими городами, то достаточно добавить всего одно правило

```
    situ(X, "Европа") :-
        situ(X, "Франция").
```

и можно будет по-прежнему использовать все остальные факты о городах Европы.

В ПРОЛОГе можно использовать составные объекты. Составные объекты позволяют описывать иерархические структуры, в которых описание одного предиката включает в себя описание других предикатов. Например,

```
implement main
    open core, console

domains
  personal_library = book(title, author, publication).
  publication = publication(publisher, year).
```

```
  collector, title, author, publisher = symbol.
  year = integer.

class predicates
    collection : (collector, personal_library) nondeterm anyflow.
clauses
    collection("Иванов", book("Война и мир", "Лев Толстой", publication
   ("Просвещение", 1990))).
```

При описании правил часто возникает необходимость использовать логические связки И и ИЛИ. В качестве связки И традиционно используется запятая, а в качестве связки ИЛИ — точка с запятой. Например,

```
gigant(X) :- rost(X, Y), Y>200.
star_or_mlad(X) :- X>70; X<10.
```

ПРОЛОГ имеет большое количество встроенных предикатов, т.е. предикатов, определяемых автоматически. Например, встроенный предикат `nl` вызывает перевод строки, а встроенный предикат `write` применяется для вывода информации на экран. Встроенные предикаты используются так же, как и определяемые пользователем предикаты, но встроенный предикат не может являться головой правила или появляться в факте. Часто используемыми встроенными предикатами являются `=` (унификация) и логическое отрицание `not`. Например,

```
student(X) :- X="Петров"; X="Иванов".
xor_student(X) :- not(X="Петров"), not(X="Иванов").
planeta(X) :- X<>"Солнце".
```

Иногда бывает полезно использовать предикаты, про которые заранее известно, истинны они или ложны. Для этих целей используют предикаты `true` и `fail`. Предикат `true` всегда истинен, в то время как `fail` всегда ложен. Последний предикат используется для управления процессом решения задачи на ПРОЛОГе.

Рассмотрим, каким образом на ПРОЛОГе можно описать задачу о семейных отношениях. Пусть имеются следующие факты:

— Иван — отец Игоря.

— Иван — отец Семёна.

— Семён — отец Лизы.

Составим программу для ответа на вопрос «Есть ли братья у Игоря?».

Формализуем имеющиеся данные. Для этой цели введём Три предиката:

— father(x, y), означающий, что x — отец y,

— man(x), означающий, что x — мужчина,

— brother(x, y), означающий, что x — брат y.

Справедливы, очевидно, следующие правила:

— Все отцы — мужчины.

— Если у детей один отец, то они единокровны.

— Брат — это единокровный мужчина.

Составим программу на языке ПРОЛОГ:

```
implement main
    open core, console

domains
    person = symbol.

class predicates
    father : (person, person) nondeterm anyflow.
    man : (person) nondeterm anyflow.
    brother : (person, person) nondeterm anyflow.

clauses
    father("Иван", "Игорь").
    father("Иван", "Семён").
    father("Семён", "Лиза").
    man(X) :-
        father(X, _).
    brother(X, Y) :-
        father(Z, Y),
        father(Z, X),
        man(X),
        X <> Y.
    run() :-
        foreach brother("Игорь", X) do
            write(X),
            nl
        end foreach,
        _ = readLine().

end implement main

goal
    mainExe::run(main::run).
```

После исполнения система выдаст `X = "Семён"`. Это отвечает нашим представлениям о правильном решении.

Приведённые примеры примитивны, но они позволяют представить полезность решений, которые может сгенерировать ПРОЛОГ при большом количестве фактов и правил в сложной предметной области. Помимо рассмотренных возможностей в ПРОЛОГе также можно осуществлять арифметические действия, организовывать рекурсию, создавать динамическую базу знаний и др. Заинтересовавшимся читателям, а также тем, кто планирует посвятить свою жизнь сфере искусственного интеллекта, рекомендуется изучить этот язык бо-

лее подробно.

Упражнение 1. Имеется следующая программа на ПРОЛОГ'е:

```
% Efremov Evgenii

implement main
    open core, console

domains
    name = string.
    counter = integer.

class facts
    закуска : (name).
    мясо : (name).
    рыба : (name).
    десерт : (name).
    калории : (name, counter).

class predicates
    блюдо : (name) nondeterm anyflow.
    обед : (name, name, name) nondeterm anyflow.
    калорийность : (name, name, name, counter) nondeterm anyflow.
    сбалансированный_обед : (name, name, name) nondeterm anyflow.

clauses
    закуска("артишоки в белом соусе").
    закуска("трюфели в шампанском").
    закуска("салат с яйцом").
    мясо("говяжье жаркое").
    мясо("цыплёнок в липовом цвете").
    рыба("окунь во фритюре").
    рыба("фаршированный судак").
    десерт("грушевое мороженое").
    десерт("клубника со взбитыми сливками").
    десерт("дыня–сюрприз").
    калории("артишоки в белом соусе", 150).
    калории("трюфели в шампанском", 212).
    калории("салат с яйцом", 202).
    калории("говяжье жаркое", 532).
    калории("цыплёнок в липовом цвете", 400).
    калории("окунь во фритюре", 270).
    калории("фаршированный судак", 254).
    калории("грушевое мороженое", 223).
    калории("клубника со взбитыми сливками", 289).
    калории("дыня–сюрприз", 122).
    блюдо(Y) :-
        мясо(Y)
        or
```

```
47          рыба(Y).
48      обед(X, Y, Z) :-
49          закуска(X),
50          блюдо(Y),
51          десерт(Z).
52      калорийность(X, Y, Z, S) :-
53          калории(X, SX),
54          калории(Y, SY),
55          калории(Z, SZ),
56          S = SX + SY + SZ.
57      сбалансированный_обед(X, Y, Z) :-
58          обед(X, Y, Z),
59          калорийность(X, Y, Z, S),
60          S < 800.
61
62  end implement main
63
64  goal
65      mainExe::run(main::run).
```

Определите, на какой вопрос отвечает программа, если заданы следующие цели:

```
    run() :-
        сбалансированный_обед("артишоки в белом соусе", "фаршированный
   судак", "клубника со взбитыми сливками"),
        fail.

    run() :-
        _ = readLine().
```

```
    run() :-
        сбалансированный_обед(X, Y, Z),
        write(X, " ", Y, " ", Z),
        nl,
        nl,
        fail.

    run() :-
        _ = readLine().
```

```
    run() :-
        сбалансированный_обед(X, Y, Z),
        X = "артишоки в белом соусе",
        Y = "говяжье жаркое",
        Z = "грушевое мороженое",
        _ = readLine().
```