Интеллектуальные системы и технологии

Лекция 3.

Логика предикатов 1-го порядка. Язык программирования Prolog

Логические

- **Логика предикатов 1-го порядка**
- Модальные логики
- Нечеткие логики
- Псевдофизические логики
- Дескрипторная логика

Логика предикатов 1-го порядка.

Формальная (логическая) система

$$S=\langle B,F,A,R\rangle,$$

где: B – алфавит,
F – формулы-факты,
A – формулы-аксиомы,
R – правила-вывода.

Логика предикатов 1-го порядка

$F(x_1, x_2 \ \ldots \ x_n)$ - предикат (логическая функция),

x_i - переменная предметной области,

n - арность предиката.

$f(x_1, x_2 \ \ldots \ x_m)$ - функция, определенная на области определения x_i.

Логика предикатов 1-го порядка

Формула состоит из предикатов, логических связок &, V, $\neg$, $\rightarrow$ и кванторов всеобщности $\forall$ и существования $\exists$

$\rightarrow$ Импликация

$F_1(x_1) \rightarrow F_2(x_2)$ Из истинности $F_1(x_1)$ следует истинность $F_2(x_2)$.

Логика предикатов 1-го порядка

$(\forall x)(F(x))$ Для всех x предикат $F(x)$ истинен

$(\exists x)(F(x))$ Существует хотя бы одно такое значение x, при котором предикат $F(x)$ истинен

$(\forall x)(F(x, y))$ x - связанная, y - свободная переменные

Логика предикатов 1-го порядка

Интерпретация

F(x) - свойство объекта x,

зеленый(кузнечик), высокий(столб),

F(x,y) - отношение между объектами x и y,

отец(Иван, Петр), учится(Иванов, НГТУ),

над(облако, земля), выше(башня, дерево),

на_территории(Россия, Байкал).

- **Исчисление предикатов первого порядка**
  - применяется
    - в диагностических и советующих ЭС
    - в системах компьютерного перевода текстов
    - для реализации символьных преобразований
      - аналитическое решение уравнений
      - аналитическое упрощение выражений
      - аналитическое интегрирование и дифференцирование и т.п.
    - в качестве метаязыка
      - в системах, требующих определения специализированных формальных систем для представления специфических знаний, например, для описания протоколов
  - программная реализация
    - непроцедурный язык программирования Prolog
    - оболочки ЭС
  - автоматизация обучения проблематична
    - как правило, формализация знаний выполняется человеком – инженером по знаниям

Логическое программирование

- Программа может быть записана при помощи логических формул, играющих роль аксиом
- Ее выполнение представляет собой доказательство формулы-запроса
- Программа – база знаний, описание отношений между понятиями и фактами, а формула-запрос – запрос к этой базе знаний
- Для решения задачи вместо алгоритма логические формулы

Alain Colmerauer

- Французский информатик
- Создатель языка Prolog

Язык Prolog

- Разработан в 1972 г.
- **Распространенные реализации:** GNU Prolog, PDC-Prolog, Quintus, SICStus, SWI-Prolog, YAP
- **Диалекты:** *ISO Prolog, Edinburgh Prolog*
- **Преемники: Visual Prolog, Mercury, Oz, Erlang, Strand**

Логическое программирование

- было предложено **John McCarthy** в 1958 в виде Advice Taker (программа, демонстрирующая логические рассуждения с использованием здравого смысла – common sense).
- **J. Alan Robinson** придумал алгоритм резолюции и унификации для логической дедукции (1963).Процедура является простой или легко программируется. Однако, наивная реализация приводит к «комбинаторному взрыву» или зацикливанию.
- 1974, **Robert Kowalski** предлагает представлять логические выражения в виде формул или ***замыканий Хорна***

  (выражений в виде правил: "if *p* then *q*"), которые сокращают логический вывод до прямой или обратной цепочки. Это упростило проблему.
- Логика может использоваться не только для представления знаний и доказательства, но и для решения таких задач как *планирование* или *обучения* (с использованием индуктивной логики).

Доказательство

- Методом резолюции Робинсона (в 1965) – методом от противного
- Целевая формула инвертируется и доказывается, что она не совместима с базой знаний (предикатами и правилами)
- В Прологе этот метод реализован в виде *унификации*

Особенности логического программирования на Прологе

- Программирование на Прологе состоит в определении отношений и в постановке вопросов, касающихся этих отношений.
- Программа состоит из предложений (clause). Предложения бывают трех типов: *факты*, *правила* и *вопросы*.
- Отношение может определяться *фактами*, перечисляющими *n*-ки объектов, для которых это отношение выполняется, или же оно может определяться *правилами*.
- *Процедура* - это множество предложений об одном и том же отношении.
- *Вопросы* напоминают запросы к некоторой базе данных. Ответ системы на вопрос представляет собой множество объектов, которые удовлетворяют запросу.
- Процесс, в результате которого пролог-система устанавливает, удовлетворяет ли объект запросу, часто довольно сложен и включает в себя логический вывод, исследование различных вариантов и, возможно, *возвраты*. Все это делается автоматически самой пролог-системой и по большей части скрыто от пользователя.
- Различают два типа смысла пролог-программ: декларативный и процедурный. Декларативный подход предпочтительнее с точки зрения программирования. Тем не менее, программист должен часто учитывать также и процедурные детали.

Структура программы на PDC-Prolog. Программа состоит из разделов:

- DOMAINS – описание доменов или типов переменных (аналог описания типов переменных)
- DATABASE – описание предикатов-фактов, которые могут добавляться/удаляться (аналог описания переменных)
- PREDICATES – описание шаблонов предикатов, встречающихся в программе
- CLAUSES – постоянная часть программы, предикаты-факты и правила (аналог исполняемой части программы)
- GOAL – целевой предикат/правило

CLAUSES

- Все предложения заканчиваются точкой
- Конъюнкция – запятая
- Дизъюнкция – точка с запятой
- Импликация – двоеточие и тире
- Рекомендуется сначала перечислять предикаты-факты, а затем правила

CLAUSES (2)

- Атом
  - Идентификатор переменной, например, F1, Person, Block1 и т.д.
  - Число
  - Символ (например, f, s и т.д.)
- Предложение (clause)
  - Предикат, например,
  - parent(a,b).
  - Правило, например,
  - brother(A, B):-male(A), brother(B,A).

Аргументы предикатов

- Аргументами предикатов в разделе CLAUSES могут быть
  - неконкретизированные (неозначенные) переменные или (имя начинается с большой латинской буквы)
  - Константы (если текст, то начинается с маленькой латинской буквы, если русские буквы, то в кавычках)
  - Анонимные (без идентификатора) переменные (знак подчеркивания) – переменные, значение которых не используется, а важно только их наличие

Пример 1 программы на PDC-Prolog

```
PREDICATES
bird(symbol)
parent(symbol,symbol)
CLAUSES
bird(sparrow).                    // Воробей – это птица.
bird(X):–parent(Y,X), bird(Y).    // X – это птица,
                                  //если у    него есть родитель,
                                  //который является птицей.
parent(sparrow,nestling).         // Воробей – родитель
                                  // птенца.
```

Унификация в Прологе

- Унификация – сопоставление
- Целевой предикат сопоставляется или нет с каким-либо предикатом-фактом или правилом (левой его частью)
- Процесс унификации ветвящийся
- На каждом уровне унификации унифицируется некоторый текущий целевой предикат. Если не удается, происходит возврат на ближайшую сверху развилку и попытка унификации с другим фактом или правилом из существующих в программе (поиск с возвратом)
- Результат унификации (доказательства) предиката – значение Yes или No, а также, значения его неконретизированных аргументов, если они есть

Унификация в Прологе (2). Сравнение двух предикатов

- В основе лежит унификация двух предикатов в следующей последовательности:
  - Сравнение имен, если совпадают, то
  - Сравнение количества аргументов, если равны, то
  - Сравнение попарно аргументов
- Сравнение пары аргументов:
  - Если один - неконкретизированная переменная, другой – конкретизированная (константа), то переменная конкретизируется (становится константой)
  - Если оба – константы, то они сравниваются на совпадение,
  - Если оба – неконретизированные переменные, то с этого момента они становятся одно и той же неконкретизированной переменной (с одним идентификатором)
- Если на каком-то уровне унификации текущий целевой предикат не доказывается (fail), то все результаты унификации, которые были получены в ходе его доказательства, отменяются

Унификация в Прологе (3). Унификация предложений

- Цели должны быть доказаны по порядку, слева, направо.
- Для доказательства некоторой цели предложения просматриваются в том порядке, в каком они появляются в тексте программы.
- Для того, чтобы доказать головную цель правила, необходимо доказать цели в теле правила. Тело правила состоит, в свою очередь из целей, которые должны быть доказаны.
- Цель считается доказанной, если с помощью соответствующих фактов доказаны все цели, находящиеся в листьевых вершинах дерева целей.

Пример 2 унификации цели

PREDICATES
little (symbol)
middle (symbol)
big (symbol)
strong (symbol)
powerful (symbol)
CLAUSES
little (cat).
little (wolf).
middle (tiger).
middle (bear).
big (elephant).
big (hippopotamus).
strong (tiger).
powerful (Animal):- middle (Animal), strong (Animal).
powerful (Animal):- big (Animal).

Goal: powerful (Animal).
CALL: powerful (_)
CALL: middle (_)
RETURN: *middle ("tiger")
CALL: strong ("tiger")
RETURN: strong ("tiger")
RETURN: *powerful ("tiger")
REDO: middle (_)
RETURN: middle ("bear")
CALL: strong ("bear")
FAIL: strong ("bear")
REDO: powerful (_)
CALL: big (_)
RETURN: *big ("elephant")
RETURN: powerful("elephant")
REDO: big (_)
RETURN: big("hippopotamus")
RETURN: powerful ("hippopotamus")

Управление поиском с возвратом

- Предикат fail, включающий поиск с возвратом. Приводит к неудаче унификации. Всегда возвращает *No*
- Предикат ! (этот предикат еще называют «отсечение»), предотвращающий поиск с возвратом. Ликвидирует возможность возврата к ближайшей развилке и попытки продолжения унификации

Пример 3 программы на PDC-Prolog

```
DOMAINS
name=symbol
PREDICATES
father (name, name)
everybody
CLAUSES
father (“Павел”, “Петр”). mother(“Лена”, “Петр”).
father (“Петр”, “Михаил”). mother(“Лена”, “Михаил”).
father (“Петр”, “Иван”).
everybody:- father (X, Y), write (X, “это отец ”, Y, “a”), nl, fail.
everybody:-mother(X,Y), write (X, “это мать”, Y, “a”), nl, fail.
GOAL
everybody.
```

Пример 4 рекурсивной программы на PDC-Prolog

```
PREDICATES
factorial (integer, real)
CLAUSES
factorial (0, 1):-!.        //факториал 0! равен 1 (граничное условие,
                            // останавливающее рекурсию)
factorial (N, FactN):-M=N-1, factorial (M, FactM), FactN= FactM*N.
        //факториал n! равен (n-1)!*n (рекурсивное условие)
Goal: factorial (3, FactN)
FactN=6
```

Процедурные знания в PDC-Prolog

- Это встроенные функции:
  - Для ввода-вывода,
  - Для добавления или удаления фактов в DATABASE
  - Для управления системой,
  - Для отладки,
  - Для преобразования данных
  - Для управления строками
- Но они выглядят и выполняются (унифицируются) так же как предикаты и возвращают значения Yes или No

Еще есть группы операторов:

- Для работы с экраном
- Для работы с внешней базой данных
- Для машинной графики
- Для редактирования текста (встроенный редактор)
- Системные (связь с ОС)
- Машинного уровня (например, работа с портами)
- Для работы с ошибками
- Для работы с сообщениями

Операторы работы с текстом программы (в PDC=Prolog и в SWI-Prolog)

consult(OSFileName) (имя файла) - (i)	- загрузка программы из файла
save(OSFileName) (имя файла) - (i)	- сохранение программы в файле
assert(Term) (предикат) - (i)	- добавление в конец
asserta(Term) (предикат) - (i)	- добавление в начало
assertz(Term) (предикат) - (i)	- добавление в конец
retract(Term) (предикат) - (i)	- удаление
retractall(Term) (предикат) - (_)	- удаление всех подходящих

Работа со списками в Прологе

- Список – это объект, который содержит конечное число других объектов. Список в Prolog’е можно приблизительно сравнить с массивами в других языках, но для списков нет необходимости заранее объявлять размерность.
- Список в Prolog’е заключается в квадратные скобки и элементы списка разделяются запятыми. Список, который не содержит ни одного элемента, называется пустым списком.
- Примеры списков:

список, элементами которого являются целые числа

[1, 2, 3]

список, элементами которого являются символы

[one, two, three]

список, элементами которого являются строки

[“One”, “Two”, “Three”]

пустой список

[]

Работа со списками в Прологе (2)

- Список является рекурсивным составным объектом, состоящим из двух частей. Составные части списка:

1. Голова списка – первый элемент списка;
2. Хвост списка – все последующие элементы, являющиеся, в свою очередь списком.

- Примеры голов и хвостов списков:

[1, 2, 3] голова списка – 1, хвост списка – [2, 3]

[1] голова списка – 1, хвост списка – [ ]

[ ] пустой список нельзя разделить на голову и хвост

- В Prolog’е используется специальный символ для разделения списка на голову и хвост –вертикальная черта |.

Пример 5 программы на PDC-Prolog

```
DOMAINS
integerlist=integer*
PREDICATES
printlist (integerlist)
CLAUSES
printlist ([]):-!.                          //для пустого списка ничего
                                            //не делать
printlist ([H|T]):-write (H), nl, printlist (T).   //для непустого
                                            // списка отделить голову,
                                  //напечатать ее, продолжить печать для
                                  //хвоста списка
```

Пример 6 на PDC-Prolog. Инверсия списка.

```
domains
    int=integer
    str=string
    strp=str*
predicates
    revers(strp,strp).
/* инверсия списка (вызываемая функция)
параметры:
    strp - исходный список
    strp - инвертированный список */
clauses
revers(X,Y):- revers([],X,Y).
revers(Y,[],Y).
revers(X1,[Z|X2],Y):- revers([Z|X1],X2,Y).
```

Недостатки логики предикатов 1-го порядка

- монотонность логического вывода, т.е. невозможность пересмотра полученных промежуточных результатов (они считаются фактами, а не гипотезами);
- невозможность применения в качестве параметров предикатов других предикатов, т.е. невозможность формулирования знаний о знаниях;
- детерминированность логического вывода, т.е. отсутствие возможности оперирования с нечеткими знаниями.