Логическое программирование

Занятие 5 / 23.05.22

Многошаговый итерационный процесс

- Процесс, состояние которого на текущем шаге зависит от состояния этого же процесса на предыдущих k шагах

Начальные условия:

$$y_0 = a_0, y_1 = a_1, \dots, y_{k-1} = a_{k-1}$$

Текущий момент:

$$y_n = f(y_n, y_{n-1}, \dots, y_{n-k}, \boldsymbol{X})$$

Условие окончания:

$$n = N$$

Одношаговый итерационный процесс

$$y_n = f(y_{n-1}, \boldsymbol{X})$$

- Вычисление суммы
- Вычисление произведения
- Вычисление факториала
- Вычисление чисел Фибоначчи

Рекурсия

- Способ описания объектов, данных, процессов или функций через самих себя
- В Prolog для создания рекурсивного определения необходимо реализовать предикат, который обращался бы к самому себе

```
ancestor(P, A) :- parent(P, A).
ancestor(P, A) :- parent(P, Parent), ancestor(Parent, A).
```

Задачи

- Сумма чисел от 1 до N
- Поиск n-го числа Фибоначчи

Рекурсивные определения

- Состоят из 2-х частей:
  - Граничные условия (начальные и окончания)
  - Рекуррентные соотношения

Факториал:

- `0! = 1` (начальное условие)
- `k = N` (условие окончания)
- `k! = k (k-1)!` (рекуррентное соотношение)

Рекурсивные определения

- В Prolog определение рекурсивного предиката разбивается на 2+ утверждений, из которых первое (первые) описывает **терминальную ситуацию** (одно из граничных условий), а последующие – **рекуррентное соотношение**, то есть предикат обращается к самому себе в теле правила
- Утверждения, описывающие терминальные ситуации, обычно заканчиваются отсечением !
- Рекуррентные соотношения могут иметь условия применимости в качестве первой цели в теле правила

Рекурсивные определения

```
class predicates
   factorial : (integer N, integer F [out]) determ.
clauses
   factorial(0, 1) :- !.
   factorial(N, N * F) :- N > 0, factorial(N – 1, F).
```

Нисходящая рекурсия

- При нисходящей рекурсии описание процесса начинается с конца (с момента N), и номер шага постепенно уменьшается на 1, пока не будет достигнута терминальная ситуация
- Смотри предыдущий пример с факториалом

Нисходящая рекурсия

- **Прямой ход** – от ввода целевого утверждения до терминальной ситуации
- **Терминальная ситуация** – одно из граничных условий
- **Обратный ход** – от терминальной ситуации до достижения вершины доказательства

Нисходящая рекурсия

- Простота реализации
- Меньше аргументов предикатов
- Требует больше памяти
- Длительнее по времени (обычно)

Восходящая рекурсия

- Процесс начинается с граничного условия, k = 0, и номер шага постепенно увеличивается, пока не будет достигнуто k = N.
- Параметры, характеризующие состояние процесса, вычисляются на каждом шаге и передаются дальше

Восходящая рекурсия

- Сложнее в реализации
- Больше аргументов предикатов
- Требует меньше памяти
- Быстрее по времени (обычно)

Задачи

- Восходящий факториал
- Восходящие числа Фибоначчи
- Возведение числа в целочисленную степень (нисходящая)
- Возведение числа в целочисленную степень (восходящая)

Перебор в пространстве состояний

- Задачи решаются обычно рекурсивно
- Необходимо определить понятие состояния
- Необходимо определить возможности перехода между состояниями
- Основную работу осуществляет рекурсивный предикат поиска пути на графе состояний (обычно – поиск в глубину, поскольку он нативно поддерживается механизмом вывода Prolog)

Перебор в пространстве состояний

- Задача о ханойской башне
- Задача о ведрах
- Задача о поиске пути в бинарной матрице

Структуры

- **Структура** – единый объект, состоящий из совокупности других объектов
- Структуру необходимо рассматривать как средство описания сложного составного объекта или сложного отношения

```
студент(фио(«Иванов», «Иван»), 103220)
```

Это не предикат, а именно структура, терм

Структуры

```
Книга(37482,
      автор(«Братко», «Иван»),
      название(«Программирование на языке Prolog»),
      издательство(«Москва», «Вильямс»),
      год_издания(2004)
)
```

Структуры

```
<функтор - имя структуры>(<список аргументов - компонент>)
```

Структуры

- Структуры описываются в разделе domains
- Структуры могут быть вложены друг в друга на любую глубину
- Структуры унифицируются друг с другом по обычным правилам

```
domains
        личность = личность(фио, адрес).
        фио = фио(string Фамилия, string Имя, string Отчество).
        адрес = адрес(string Город, string Улица, integer Дом, integer Квартира).
```

Объекты с множественным типом данных

- При помощи ; в разделе domains можно описать классы объектов, состоящие из множества типов данных

```
domains
    объект = книга(string Название, автор Автор);
        бытовая_техника(string Наименование);
        собака(string Кличка);
        дом.
    автор = автор(string Фамилия, string Имя, string Отчество).
class facts
    владеет : (string Имя, объект Объект).
```