Проектиране и Тестиране на Софтуер
ТУ, кат. КС, летен семестър 2012

Лекция 51

Тема:

Логическо Програмиране (Пролог)

Разпространение на ПЕ

- Виж следния адрес:

http://www.tiobe.com/index.htm?tiobe_index

Съдържание:

- Стилове на Програмиране
- Логическо Програмиране
- Част 1
  - Основни понятия
- Част 2
  - Въведение в Пролог

Стилове на Програмиране

Дейностите в програмирането не се подчиняват на единни водещи принципи. Съществуват стилове на програмиране, които могат да се класифицират така:

- Процедурно програмиране (procedure based programming or imperative programming);
- Логическо програмиране (logic programming);
- Функционално програмиране (functional programming).

Процедурно програмиране

Красноречиво пояснение на този стил представлява заглавието на книгата на Никлаус Вирт

Алгоритми + Структури от Данни = Програми

Програмистът съставя програма и указва на компютъра ***как?*** на базата на избран алгоритъм и възприети структури от данни да се намери решение на поставената задача. Типични процедурни ПЕ са Fortran, Algol, PL/1, Basic, Pascal, C/C++/C#, Java.

Логическо програмиране

Водещият принцип в този стил гласи

Правила + Факти = Програми

Изпълнението на една програма от ЛП се свежда до *доказателство* или *извод* на определен факт, правило или твърдение чрез сканиране на предварително изградена БД, съдържаща правила и факти от определена предметна област. Алгоритмичният въпрос от процедурното програмиране ***как*** да се реши една задача се измества от въпроса ***какво?*** *(каква задача)* има да се решава и как да се опише. Този стил е известен и като декларативно програмиране. Език за ЛПе Пролог (Prolog = PROgramming in LOGic).

Функционално програмиране

Водещият принцип в този стил гласи

Програма = Композиция от Функции

Наименованието на стила подсказава важността на обекта *функция*. Системите за ФП предоставят на програмиста множество от базови функции, както и средства за дефиниране на нови функции от базовите, с което се постига решаване на поставения проблем. Този стил е известен още и като апликативно програмиране. Език за ФП е Лисп (Lisp – LISt Programming).

Стилове на Програмиране

Imperative (procedure driven) programming

Algorithms + Data Structures = Programs

Structured Programming: **Thinking in Functions!**

Classes + Objects = Programs

OOP: **Thinking in Objects!**

Logic programming

Rules + Facts = Programs

Functional programming

Composition of Functions = Programs

Логическо Програмиране

Процедурно програмиране с/у логическо прграмиране

Алгоритмичният въпрос **КАК**:

HOW to solve a problem?

Се заменя с по-общата задача **КАКВО**

WHAT PROBLEM is to be solved?

Инструменталният език за лог. програмиране е ***ProLog – PROgramming in LOGic***

Част 1

Основни понятия

Съдържание:

- Предикатно смятане
  - Предикат
  - Клауза (клаузна форма)
  - Резолвента
  - Метод на резолюцията

Основни понятия

В своята същност логическото програмиране се основава на принципите на формалната логика и предикатното смятане. Съществен смисъл в този аспект имат понятията за предикат, клаузна форма, резолвента, метод на резолюцията.

Част 2

Въведение в Пролог

Съдържание:

- Увод
- Факти
- Цели (въпроси)
- Променливи в Пролог
- Конюнкции, дизюнкции и отрицание
- Правила
- Рекурсия в Пролог

Увод

Същината на ЛП е в деклариране (задаване) на факти и правила, които описват определена предметна област. Фактите и правилата формират БД. Към базата данни се задават въпроси. Няма програмиране с избор на алгоритъм и структури данни. Една Prolog-програма е съвкупност от данни (факти) за обекти и отношения между обекти. Изпълнението на една програма Prolog се състои в това потребителят да формулира цел (goal) – въпрос относно фактите и правилата в БД. Целта се задава и системата се опитва да я удовлетвори.

Въведение

Доколкото програмите тук не съдържат алгоритми и структури данни, специалистите по ЛП не са класически програмисти. Те са специалисти по знания в определена област (knowledge engineers), които проектират и реализират практически системи за логическо програмиране.

Въведение

Съставянето на една програма на Prolog включва:

- задаване на факти за обекти и отношения,
- дефиниране на правила за множество факти и отношения.

Изпълнението на една Prolog програма включва задаване на въпроси относно описаните факти и дефинираните правила.

Факти

Фактите в Пролог служат за задаване на безусловно верни същности – свойства и отношения. Пролог възприема един факт за верен независимо дали той е верен в реалния свят. Фактът се счита за верен, т.к. програмистът е заявил така и това е в сила за БД, която е съставена и се управлява от системата за ЛП.

Факти. Примери

Ето факт като изречение на естествен език:

John likes Mary.

Това е факт с два обекта: *John, Mary*

и едно отношение: *likes*

Същият факт представен на Пролог:

likes(*john, mary*).

likes се нарича предикат.

john, mary се наричат аргументи.

Предикатът и аргументите се пишат с малки букви.

Всеки факт се затваря с терминален символ точка.

Факти. Примери

Изречение на естествен език: *John is a student*.

обект: John

отношение: статус на студент

Факт на Пролог: *student(john)*.

Фраза на естествен език: *Nature*

0 обекти, 1 отношение

Факт на Пролог: *nature*.

Факти. Примери

Налице са вградени предикати за рутинни процедурни действия като В/И, комуникация с ОС и други.

Например, предикатът *write* служи за извеждане на екран.

write('Hello').

извежда текста 'Hello' на екрана.

Как се пишат факти?

- Отношения и обекти с малки букви.
- Всеки факт завършва с точка.
- Отношенията се наричат предикати.
- Обектите се наричат аргументи.
- Допуска се произволен брой аргументи.
- Факти се интерпретират от програмиста.
- Колекция (съвкупност) от факти е БД.

Цели (въпроси)

Процесът на извод (доказателство) в Пролог =
Изпълнение на една ЛП програма;

Процесът на извод в Пролог включва:
удовлетворяване на цел (подцел);
пре-удовлетворяване на цел (подцел);
механизъм на възврат (backtracking).

След съставяне на БД с факти, потребителят може да задава въпроси, свързани с базата данни. Въпросите се наричат цели (***goals)*** или заявки ***(queries)***.

goal:- likes(john,mary).
Yes
goal:- likes(john,violet).
No
goal:- student(john).
Yes
goal:- student(peter).
No

Отговор Yes значи вярно или доказуемо (изводимо) при сканиране на БД.

След съставяне на БД с факти, потребителят може да задава въпроси, свързани с базата данни. Въпросите се наричат цели (***goals)*** или заявки ***(queries)***.

goal:- likes(john,mary).
Yes
goal:- likes(john,violet).
No
goal:- student(john).
Yes
goal:- student(peter).
No

Отговор Yes значи вярно или доказуемо (изводимо) при сканиране на БД.
Отговор No значи невярно или недоказуемо (неизводимо) при сканиране на БД.

Факт и въпрос (goal) съвпадат (are unified), ако:

- Имената на предикатите са еднакви;
- Съответните аргументи също са еднакви.

Променливи

Променивите служат за задаване на абстрактни (по-общи) въпроси. Прилагат се, когато не можем или не искаме да назовем даден обект.

Въпрос на естествен език: Who is a student?

Цел на Пролог: goal:- student(X).

X = john

Пролог сканира фактите в БД, за да намери обект, който да бъде съпоставен на променливата X.

Променливи

Въпрос на ест. език: Кого харесва Джон?

Who does John like?

Цел на Пролог: goal:- likes(john, X).

X = mary

Пролог сканира фактите в БД, за да намери обект, който да бъде съпоставен на променливата X.

- X е променлива.
- Променливите се пишат с голяма начална буква.
- Променливите могат да имат две състояния:
  - Свободна
  - Свързана

Факт и въпрос (goal) съвпадат (are unified), ако:

- Имената на предикатите са еднакви;
- Съответните аргументи също са еднакви, или поне един от неидентичните аргументи е свободна променлива.

- Пролог отговаря No, ако факт и въпрос не съвпадат.
- В случай на успех, Пролог отговаря подобно на X = mary.
- След такъв отговор има две възможности:

A/ CR (carriage return или Enter). Това значи – отговорът удовлетврява.

B/ ;CR. Това значи – отговорът не удовлетворява, търсенето в БД следва да продължи от текущата позиция, като всички свързани променливи стават свободни.

Казва се, че Пролог пробва да пре-удовлетвори целта.

Други примери

Facts:

student(peter).

student(john).

student(mary).

Goals:

goal:-student(X).

X = peter ;cr

X = john ;cr

X = mary ;cr

No

След ;cr Пролог освобождава свързаната променлива X. X се възстановява като свободна променлива. X забравя обекта, с който е бил свързан.

Още един пример

Facts:

```
lives_in(john,sofia).
lives_in(peter,varna).
lives_in(mary,plovdiv).
lives_in(dimo,rousse).
```

Goals:

```
goal:-lives_in(Person,City).
Person = john
City = sofia     ;cr
Person = peter
City = varna     ;cr
Person = mary
City = plovdiv ;cr
Person = dimo
City = rousse   ;cr
No
```

Конюнкции

Въпроси за сложни отношения се задават като последователност от цели, разделени с , (commas)

Пример: Do John and Mary like each other?

goal:likes(john,mary) , likes(mary,john).

Пример: Is there anything that both John & Mary like?

goal: likes(john,X) , likes(mary,X).

Въпросът е конюнкция от цели. Пролог опитва да удовлетвори всички цели от ляво на дясно.

Ако всички цели са удовлетворени, конюнкцията успява. В противен случай, тя пропада.

факти: student(john).

student(peter).

student(mary).

lives_in(peter,varna).

lives_in(john,sofia).

lives_in(mary,plovdiv).

Въпрос: goal:- student(X) , lives_in(X,varna)

Два процеса са активни: пре-удовлетворяване на първата подцел, и удовлетворяване на втората подцел.

Това е механизъм с възврат BACKTRACKING.

Отговорът е: X = peter

Правила

Правила се ползват, когато един факт зависи от друг факт или от група факти.

Пример: John likes anyone who likes coke.

likes(john,X) :- likes(X,coke).

Общ формат на правило:‘:-’ значи ‘ако’

заглавие :- тяло

Лява част :- дясна част

Правила се ползват за дефиниции.

Пример 1: птицата е животно с пера.
A bird is an animal with feathers.

X е птица, ако X е животно и X има пера.

X is a bird, if X is an animal and X has feathers.

```
bird(X) :- animal(X) , has(X,feathers).
```

Правила се ползват за дефиниции.

Пример 2: X е брат на Y, ако X е мъж, и двамата X и Y имат общ родител

X is Y's brother, if X is male and both X and Y have the same parent.

```
brother(X,Y) :- male(X),parent(Z,X),parent(Z,Y).

sister(X,Y) :- female(X),parent(Z,X),parent(Z,Y).
```

Summary 2

- Пролог клаузите са от 3 типа: факти, правила и въпроси-цели (goals).
- Фактите декларират свойства и отношения, които винаги са верни.
- Правилата декларират същности, които зависят от други условия.
- Задавайки въпроси, потребителят проверява дадена същност дали е вярна.
- Commas, разделящи под цели, са конюнкции.
- Semicolons, разделящи под цели, са дизюнкции.

Рекурсия в Пролог

Правилата служат за създаване на дефиниции, които се базират на други съществуващи факти.

Рекурсивните правила се прилагат, когато се дефинират нови отношения на основата на същия тип отношения.

Рекурсивни правила - примери

Конфигурация кубове k1, k2, k3, k4, k5, k6

Факти:

is_over(k2, k1).

is_over(k4, k3).

is_over(k5, k4).

is_over(k6, k5).

Рекурсивни правила - примери

Правило, описващо понятие кула (tower) чрез релацията върху (is_over):

Кула с Връх k2, Основа k1

```
tower(k2, k1) :- is_over(k2, k1).
```

Кула с Връх k4, Основа k3

```
tower(k4, k3) :- is_over(k4, k3).
```

Кула с Връх k5, Основа k3

```
tower(k5, k3) :- is_over(k5, k4) , is_over(k4, k3).
```

Кула с Връх k6, Основа k3

```
tower(k6, k3) :- is_over(k6, k5) , is_over(k5, k4) , is_over(k4, k3).
```

Рекурсивни правила - примери

Правило, описващо понятие кула (tower) чрез релацията върху (is_over):

Кула с Връх Z, Основа X

```
tower(Z, X) :- is_over(Z, X).
```

Кула с Връх Z, куб Y1, Основа X

```
tower(Z, X) :- is_over(Z, Y1) , is_over(Y1, X).
```

Кула с Връх Z, кубове Y2, Y1, Основа X

```
tower(Z, X) :- is_over(Z, Y2) , is_over(Y2, Y1) , is_over(Y1, X).
```

Кула с Връх Z, кубове Y3, Y2, Y1, Основа X

```
tower(Z, X) :- is_over(Z, Y3) , is_over(Y3, Y2) , is_over(Y2, Y1) , is_over(Y1, X).
```

Рекурсивни правила - примери

Най-примитивен е подходът да се дефинира кула чрез релацията *is_over* само.

Така се описват само конкретни конфигурации на кули като на предишните два слайда.

По-добър и ефективен подход е да се опише (дефинира) кула по абстрактен начин като се приложи рекурсивен тип дефиниция.

Рекурсивни правила - примери

Прост случай, гранично условие

Кула с Връх Z, основа X

```
tower(Z, X) :- is_over(Z, X).
```

Общ случай, пре-дефиниране на кула с рекурсия

Кула с връх Z, куб Y, кубове . . ., Основа X

```
tower(Z, X) :- is_over(Z, Y) , tower(Y, X).
```

Пролог БД

Факти: is_over(k2, k1).
is_over(k4, k3).
is_over(k5, k4).
is_over(k6, k5).

Правила:tower(Z, X) :- is_over(Z, X).
tower(Z, X) :- is_over(Z, Y) , tower(Y, X).

Цели (въпроси)

goal:- tower(k2, k1).

Yes

goal:- tower(k6, k3).

Yes

goal:- tower(k1, k6).

No

Цели (въпроси)

goal:- tower(X, k3).

X = k4 ;cr

X = k5 ;cr

X = k6 ;cr

No

Цели (въпроси)

goal:- tower(k6, X).

X = k5 ;cr

X = k4 ;cr

X = k3 ;cr

No

Благодаря
За
Вниманието