Prolog

Z FITwiki

Obsah

Logické programování

- logický program = libovolná konečná množina programových Hornových klauzulí
- **Hornovy klauzule** = klauzule (disjunkcia literalov) s nejvýše jedním pozitivním literálem (dají se reprezentovat jako implikace)
- odvozování (dokazování) cílů založené na SLD-rezoluci
  - rezoluce - dokazuje se nesplnitelnost formulí, rezoluční strom, v kořenu je dokazovaná formule C, v listech jsou jednotlivé klauzule S, ze kterých je možné C dokázat, ve vnitřních listech resolventy
  - **SLD rezoluce** - lineární vstupní rezoluce se selekčním pravidlem
    - selekční pravidlo = libovolná funkce, která vybere literál z každé uspořádané cílové klauzule
    - pomocí selekčního pravidla algoritmizujeme výběr literálu z cílové klauzule, na které se bude rezolvovat
    - SLD-rezoluce je korektní a úplná pro Hornovy klauzule
- deklarativní
- teoretický model zachovává úplnost

Prolog

- konkrétní implementace logického programovacího jazyka
- pro řešení problémů v oblasti umělé inteligence a zpracování přirozeného jazyka

Datové typy

termy - konstanty, proměnné, složené termy (= struktury)

- konstanty
  - celá čísla (-12)
  - desetinná čísla (1.3345)
  - atomy
    1. uzavrene v apostrofem (napr. 'Pavel')
    2. prvni pismeno male + C style identifikator pravidla (napr. pavel, sUP3R_1d3nt)
    3. pouze specialni symboly (napr. [], :-)
- proměnné - první písmeno velké (N, Result), případně anonymní proměnné - nezajímá nás hodnota (je_dite(X) :- ditetem(X,_).)
- složené termy - skládají se z:
  - funktor - jméno, arita (často použití jako operátory - $A < B$ místo $< (AB)$)
  - argumenty - například (bod(X,Y,Z), tree(Value,tree(LV,LL,LR),tree(RV,RL,RR)))

Seznamy

- rekurzivní datová struktura (složený term)
- uspořádaná posloupnost prvků
- heterogenní (Prolog netypovaný jazyk), prvky libovolné termy včetně seznamů
- pro tvorbu seznamů slouží funktor . arity 2 - zapisuje se jako ./2
- prázdný seznam []
- možnost tvorby seznamů:
  - přes funktor - .(head, body) - např: .(a, []) .(a, .(b, .(c,[])))

- přímo v [] s využitím | - [head, body], [head | body] - např: [a, b, c] [a | [b, c]]

Programy
množina programových Hornových klauzulí

- fakta
  - Hornovy klauzule bez negativního literálu
  - např. ditetem(andulka, jan).
  - vlastně se jedná o pravidla s prázdným tělem
- pravidla
  - Hornovy klauzule s aspoň 1 negativním literálem
  - např. vnukem(X,Y) :- ditetem(X,Z), ditetem(Z,Y).
  - hlava a tělo pravidla
- cíle
  - Hornovy klauzule bez pozitivního literálu
  - reprezentují dotazy – spuštění výpočtu
  - např. ?- vnukem(petr, pavel)

fakta a pravidla tvoří hypotetický základ pro důkaz

cíle jsou predikáty, které chceme z dané hypotézy ověřit, zda platí nebo ne

Vyhodnocování

- vyhodnocování se provádí na základě SLD rezoluce
- podcíle jsou expandované do hloubky (ztráta obecnosti - možné uváznutí v nekonečné větvi SLD-stromu i pokud by další větev nalezla řešení)
- podcíle jsou vybírané zleva doprava
- snaha unifikovat podcíl s hlavičkou faktu (klauzule vložené do programu), potom je tělo odpovídající nalezené klauzule procházené zleva doprava a každý atom na pravé straně je zpracovávaný do hloubky
- při selhání podcíle se vykonává návrat (backtracking) - vyhodnocovací mechanismus se snaží navrátit backtracking do místa prohledávání a znovu, pro právě selhávající (dílčí) podcíl, se snaží najít jinou hlavičku, kde by uspěl
- backtracking lze explicitně vyvolat i v případě úspěšného nalezní - potom jsou prohledávány další možnosti

Unifikace

- Unifikace je v Prologu základní a jedinou operací, která řídí vyhodnocení a díky níž se dostáváme k výsledku
- Probíhá při vyhodnocení SLD rezolucí, když zadáme cíl a potom u všech (dílčích) podcílů, nebo explicitně zadáním predikátu =, či jeho negované formy \=
- probíhá podle Robinsonova unifikačního algoritmu - nevykonává se kontrola výskytu (neodhalí se např. X = f(X), co může vést na nekonečný výsledný term)

Využití

- Přiřazení (navázání) hodnoty k nějaké proměnné: <hodnota> = <proměnná> (lze zapsat i opačně)
- Test na rovnost (unifikovatelnost), i složitých struktur: <term1> = <term2>
- Selektor položek z datových struktur/seznamů: <seznam> = [<hlavička seznamu> | <tělo seznamu>]
- Vytváření datových struktur/seznamů: <proměnná> = [1,2,3]
- Předávání parametrů hodnotou, kdy term je předán jako skutečný argument.
- Předávání parametrů odkazem, kdy je předána proměnná (volná, nenavázaná) jako skutečný argument.
- Sdílení proměnných, vytváření synonym

Vestavěné predikáty

- vlastnost - obvykle nejsou znovusplnitelné, nezbytné pro praktické programování
- Aritmetické operace - vyhodnocování pomocí operátoru **is** (<proměnná> **is** <výraz>)
  - Výraz je nejdříve vyhodnocen a poté unifikován s proměnnou vlevo od operátoru (pro X is X+1 vždy selže)
- Testování typu dat (test za běhu) - test na to, zda proměnná obsahuje/je navázána na:
  - (celé) číslo: **integer**(<proměnná>)
  - atom (základní, nestrukturovaná hodnota): **atom**(<proměnná>)
  - atom, nebo číslo: **atomic**(<proměnná>)
- Metalogické predikáty
  - test na to, zda proměnná je navázána na nějakou hodnotu, či ne (**var**(<proměnná>), **nonvar**(<proměnná>))
  - test na totožné (identické) objekty: <proměnná> == <proměnná>
- Operace s klauzulemi - práce s databází
  - Vložení klauzule, či faktu: **assert**(<klauzule>)
  - Výběr klauzule z databáze podle hlavičky: **clause**(<hlava>, <tělo>)
  - Odebrání klauzule z databáze: **retract**(<klauzule>)
  - Zpracování klauzulí ze souboru: **consult**(<soubor>)

Operátor řezu

Pekné, názorné a pochopiteľné vysvetlenie operátoru rezu je na tejto stránke: [1] (http://ksp.mff.cuni.cz/tasks/19/tasks4.html#task6) .

- právě jednou uspívající operátor
- Pokud dojde k zpětnému navracení v programu, tak dostane-li se navracení až k operátoru řezu, tak selže celý podcíl a všechny body znovuuspění, které se mezi operátorem řezu a hlavičkou klauzule vyskytly, jsou zapomenuty

- zelený řez - vliv pouze na efektivnost programu

```
max2(X,Y,X):-X>Y,!.
max2(X,Y,Y):-Y>=X.
```

- červený řez - nutný, aby program fungoval správně

```
max(X,Y,Z):-X>Y,!, X=Z.
max(X,Y,Y).
```

Správné použití

- Chceme Prologu sdělit, že již byla nalezena správná varianta podcíle a všechny ostatní již nejsou relevantní (něco jako koncová podmínka):

```
sum_to(1,1) :- !.
sum_to(N,Res) :-
  N1 is N-1,
  sum_to(N1,Res1),
  Res is Res1+N.
```

  pokud by chyběl operátor řezu, při navracení by byla využita další definice predikátu sum to, na řádku 2, čímž by se výpočet dostal na nesprávnou větev (podtečení).

- Chceme, aby cíl ihned selhal. Bylo dosaženo stavu, kdy další prohledávání nevede k řešení. Potom operátor řezu řetězíme s predikátem fail.

```
zena(X) :- muz(X), !, fail.
zena(X).
```

- Chceme ukončit generování alternativních řešení (bud nemají význam nebo nejsou správná)

```
is_integer(0).
is_integer(X) :- is_integer(Y), X is Y+1.
divide(N1, N2, Result) :-
 is_integer(Result),
 Product1 is Result*N2,
 Product2 is (Result+1)*N2,
 Product1 =< N1,
 Product2 > N1,
 !.
```

- Příklad použití operátoru řezu v negaci.

```
not(Goal) :- call(Goal),!,fail.
not(Goal).
```

Nesprávné použití

- může zamezit nalezení všech správných řešení i tam, kde to vyžadujeme nebo je možné obdržet i řešení nesprávná

```
max1(X,Y,X):-X>Y.
max1(X,Y,Y).
```

- s operátorem řezu mají predikáty silně vymezený charakter vzhledem k tomu, u kterých parametrů je možné mít volnou proměnnou a kde je nutné mít již konkretizovanou hodnotu

```
num_of_parents(adam, 0) :- !.
num_of_parents(eve, 0) :- !.
num_of_parents(X, 2).

(?- num_of_parents(eve, 2). bude yes - nesprávné)
```


Kategorie: Státnice 2016 | Státnice FPR | Funkcionální a logické programování

- Stránka byla naposledy editována 10. 6. 2016 v 15:04.