Logické programovanie

Ivan Kapustík

Vývoj

- Snahy naučiť stroj logicky uvažovať
  - Rôzne odvodzovače a dokazovače nad formálnou logikou
  - Základom je oddelenie údajov – logických formúl a odvozovacieho mechanizmu
  - Problémy s úpravou a interpretáciou logických formúl
- Prolog
  - Deklaratívny zápis vo forme logického programu
  - Využíva Hornove klauzuly, nad ktorými je možné vytvoriť jednoduchú a pritom úplnú rezolvenciu (dokazovanie)
  - Prolog očakáva dopyt a jeho odpoveďou je Yes alebo No
  - Zaujímavejšie odpovede poskytuje prostredníctvom unifikácie

Hornove klauzuly

- $\forall x_1, \ldots, x_n \; p \wedge \neg q_1 \wedge \ \ldots \ \wedge \neg q_n$
- $\forall x_1, \ldots, x_n \; p \wedge \neg(q_1 \vee \ \ldots \ \vee q_n)$
- $\forall x_1, \ldots, x_n \; p \Leftarrow (q_1 \vee \ \ldots \ \vee q_n)$

- V Prologu
- pravidlo
  - p :- $q_1, \ \ldots, q_n.$
- fakt
  - p.
- dopyt alebo cieľ
  - ?- $q_1, \ \ldots, q_n.$
- p nazývame hlava klauzuly, časť za :- je telo klauzuly

Program v Prologu

- Postupnosť klauzúl s podobnou hlavou tvorí definíciu (deklaráciu) predikátu.
- Dva prvky jazyka sú si podobné, ak existuje unifikátor, ktorý z nich vytvorí rovnaký prvok
- Majme p(X, b) a p(a, Y). Unifikátor {a/X , b/Y} z oboch vytvorí p(a, b)

- Množina faktov a definícií predikátov tvorí program v Prologu
- Vykonávanie programu sa spustí vložením dopytu, na ktorý Prolog odpovie Yes, prípadne No alebo vypísaním hodnôt premenných z dopytu, pre ktoré je dopyt nad programom dokázaný.

Pojmy jazyka

- Term je jednoduchý objekt alebo štruktúra
- Jednoduchý objekt je konštanta alebo premenná
- Konštanta je atóm alebo číslo (reťazec a i.)
- Atóm: a, atom, co_si, x3, ‘Peter’
- Premenná: Jano, V, Len, _prem, _
- Platnosť premennej je vždy len v rámci jedinej klauzuly
- Štruktúra je komplexný term, zapísaný v tvare predikátu (p(a,b,X)), v operátorovom tvare (5 + Z) alebo aj ich kombinácii. Jeden zo štruktúrovaných termov je aj zoznam.

Zoznam v Prologu

- Prázdny zoznam:
  - []
- Neprázdny zoznam:
  - [a, b, 1, [5, [], a], f(a,3)]
- Zoznam v špecifickom tvare Hlava a Telo/Chvost (Head Tail):
  - [H | T], kde H je prvý prvok a T je zvyšok zoznamu
- Zvislá čiara funguje ako selektor alebo konštruktor zoznamu (podobne ako v Elm-e), podľa toho, či sa jedná o vstupný alebo výstupný argument predikátu
- [1, 2, 3, 4] = [1 | [2, 3, 4]]

Štandardné predikáty

- Neformálny zápis často: Funktor/Arity (Názov predikátu/počet argumentov)
  - atom/1, number/1, integer/1, atomic/1, var/1, nonvar/1, halt/0
- Podrobnejší zápis (budeme používať):
- is_list(+Term)
  - +Arg – vstupný argument, musí sa vyhodnotiť na konštantu
  - -Arg – výstupný argument, musí tam byť premenná, ktorá získa hodnotu po vykonaní predikátu
  - ?Arg – argument, ktorý môže byť aj vstupný aj výstupný. Ak je tam premenná, naviaže sa na nejakú hodnotu, ak je tam konštanta, porovná sa na zhodu

Štandardné operátory

- Matematické: +, -, *, /
- Relačné: <, >, **=<**, >=
- Zhoda a podobnosť:
- = unifikácia, ľavá a pravá strana sa porovnajú a ak sú tam premenné, pokúsia sa naviazať.
- == zhoda bez naviazania premenných
- \= nezhodujú sa po unifikácii
- =\= nezhodujú sa presne
- is očakáva na pravej strane matematický výraz, ten vyhodnotí a potom funguje ako =

Rekurzia

- Na najvyššej úrovni – jednoduchý, lineárny zoznam
  - [a, b, 1, 3]

- Na ľubovoľnej úrovni – vnáraný zoznam
  - [a, b, [c, a], d, [r, 2, a(w), [4, e, [m]]], 3]
  - Zisťovanie výskytu, výmena prvkov

Backtracking – spätný chod

- Vykonávanie programu zodpovedá volaniu predikátu – `call`
- Volanie môže vrátiť úspech (`exit`) alebo neúspech (`fail`)
- V prípade úspechu program volá štandardne nasledujúci predikát.
- V prípade neúspechu program volá predchádzajúci predikát, ak taký existuje. Ale už nie cez `call` ale `redo`.
- Cez `redo` sa snaží Prolog splniť predikát iným spôsobom

Rez

- Symbol rezu je „!“
- Pri prvom volaní – `call` – je splnený
- Pri opakovanom volaní – `redo` – spôsobí nesplnenie cieľa – `fail` – na predchádzajúcej úrovni

```
sucet([],0).
sucet([H|T],Sum):-
    number(H), !, %vkladáme za vhodný test
    sucet(T, S),
    Sum is H + S.
sucet([H|T],Sum):-
    sucet(T, Sum).
```

Riadiace prostriedky

- Poradie cieľov a klauzúl
- Vetvenie – viacero klauzúl – alternatívy „;“
- Spätný chod
  - `fail, !, true`
- Cykly
  - Rekurzia – na najvyššej úrovni, vo vnáranom zozname
  - Iterácia – viacnásobne splnený predikát
    - `repeat`
- `not/1`
- `call/1, =..`
- Vstup a výstup
- Zmena údajov v databáze

Vstup a výstup

- `read(-Term), get(-Char)`
- `write(+Term), writeq(+Term), nl`

- `see(+Súbor), seen`
- `tell(+Súbor), told`

- `open(+SrcDest, +Mode, -Stream)`
  `close(+Stream)`
  `read(+Stream, -Term)`
  `write(+Stream, +Term)`

Cykly

- ```
  zena(eva).
  zena(jana).
  zena(viera).
  ```
- ```
  ?- zena(X).
  ```
- ```
  ?- zena(X), writeln(X), fail.
  ```
- ```
  ?- member(X,[a, b, c, d]), writeln(X),
  fail.
  ```

- ```
  repeat.
  repeat :- repeat.
  ```

- ```
  repeat, read(X), write(X), X = end.
  ```
  %Pozor na nekonečný cyklus!

Operátory

- `op(+Precedence, +Type, +Name)`
- Precedencia: 0 až 1200, vyššie číslo = nižšia priorita, odporúčané menej ako 1000, aby korektne fungovala „,“.
- Typ: `fx`, `fy` – unárny operátor
- `xfx`, `xfy`, `yfx` – binárny operátor
- „x“ je operand s nižšou precedenciou
- „y“ je operand s rovnakou alebo nižšou precedenciou
- Mat. operátory sú `yfx`
  - `5 + 6 + 7` zodpovedá `(5 + 6) + 7`
- „,“, „;“ sú `xfy`
  - `nl, nl, nl` zodpovedá `nl, (nl, nl)`
- `current_op(?Precedence, ?Type, ?Name)`

Konštrukcia predikátu

- `A =.. [funk, a, b]`
- `call(A)`
- `?- member(5,[a, b, c]) =..[F, A1, A2], P =..[F, A1, [3, 4, 5]), call(P).`

- Tvorba ekvivalentu funkcionálov
- Špecifická práca s argumentami
- Metaprogramovanie

Databáza

- `dynamic/1`
- `asserta(+Term), assertz(+Term)`
- `retract(+Term), retractall(+Head)`

- Nájdenie všetkých riešení
- `findall(+Template, :Goal, -Bag)`
- `bagof(+Template, :Goal, -Bag)`
- `setof(+Template, +Goal, -Set)`