Język programowania Prolog

- oparty na teoretycznych zasadach programowania w logice
- rozszerzenie o mechanizmy pozalogiczne (metalogiczne)
- operacje niskopoziomowe (arytmetyka, operacje wejścia-wyjścia)
- sterowanie

Implementacje Prologu:

– SICStus Prolog (Swedish Institute of Computer Science)
(profesjonalne oprogramowanie) [Unix, Linux, Windows]
– SWI-Prolog (Free Software, GNU License)
(`http://www.swi-prolog.org/`)
– MIM Prolog (Instytut Informatyki UW) [MS-DOS, IBM PC]

Składnia programów

- brak deklaracji (zmiennych, stałych, ...)
- nazwy zmiennych zaczynają się od wielkich liter,
nazwy stałych — od małych liter
- zmienne anonimowe (_)
- zamiast ← piszemy `:-`
- każdą klauzulę kończymy kropką
- klauzule unarne — atom zakończony kropką
- komentarze — `%` oraz `/* ... */`

Reprezentacja list

- lista pusta `[]`
- lista niepusta, k-elementowa: `[ E1, E2, ..., Ek ]`
- lista niepusta: `[ Elem | Lista ]`
  gdzie `Elem` — pierwszy element listy,
  `Lista` — dalsza część listy
- lista niepusta: `[ _ | _ ]`
- np. lista 2-elementowa

```
            [E1, E2]  lub  [ E1 | [E2] ]
    lub     [ E1 | [ E2 | [] ] ]
    lub     [ E1, E2 | [] ]
```

Przykłady

```
% member( X, L ) wiw gdy X jest elementem listy L

member( X, [ X | L ] ).
member( X, [ _ | L ] ) :-  member( X, L ).


% append( L1, L2, L ) wiw gdy L1 * L2 = L

append( [], L, L ).
append( [ Elem | L1 ], L2, [ Elem | L ] ) :-
       append( L1, L2, L ).
```

Sterowanie

- uporządkowanie klauzul (tekst programu)
- reguła obliczeniowa: wybór skrajnie lewego atomu
- przeszukiwanie w głąb

Mechanizm nawracania

• wybór klauzuli programu (zgodność nagłówka z wybranym atomem)

• jeśli nie jest to ostatnia (tekstowo) klauzula procedury, to utwórz **punkt nawrotu**

• porażka (oraz żądanie podania wszystkich odpowiedzi) powoduje **nawrót**

punkt nawrotu: zestaw informacji niezbędnych do wznowienia procesu obliczeniowego
– identyfikacja następnej klauzuli
– stan obliczenia (dowiązania zmiennych)

nawrót – brak punktów nawrotu: porażka
– cofnięcie do ostatnio utworzonego punktu nawrotu
unieważnienie wszystkich dokonanych obliczeń,
przywrócenie stanu obliczeń sprzed wyboru klauzuli,
której wybór doprowadził do porażki

efekty uboczne – nie możliwe unieważnienie niektórych dokonanych obliczeń (m.in. operacji wejścia-wyjścia)

Procedury standardowe

Operacje arytmetyczne

wyrażenie arytmetyczne: liczba całkowita oraz termy postaci:

A + B, A − B, A ∗ B
A / B (dzielenie rzeczywiste)
A // B (dzielenie całkowite)
A mod B (operacja *modulo*)

gdzie A i B są wyrażeniami arytmetycznymi

Var **is** Wyrażenie

Var – zmienna lub wyrażenie arytmetyczne,
Wyrażenie – wyrażenie arytmetyczne

Porównywanie **wartości** wyrażeń arytmetycznych

X =:= Y równość
X =\= Y różność
X < Y
X > Y
X =< Y
X >= Y

Przykład.

```
% length( L, N ) wiw gdy L jest lista N-elementowa

   length( [], 0 ).
   length( [ _ | L ], N ) :-
          length( L, M ),  N is M + 1.
```

Znakowe operacje wejścia-wyjścia

standardowy plik wejściowy i wyjściowy o nazwie `user`

get0(C) — wczytanie następnego znaku z bieżącego pliku wejściowego, uzgodnienie go z C; koniec pliku: -1 (SICStus)

get(C) — wczytanie następnego znaku różnego od spacji

put(C) — wypisanie znaku o kodzie C

nl — wypisanie znaku końca wiersza

see(Nazwa_Pliku) — otwarcie wskazanego pliku, uczynienie go bieżącym plikiem wejściowym

seen — zamknięcie bieżącego pliku wejściowego
MIM Prolog: powrót do poprzedniego (nic nie robi, gdy brak poprzednika); SICStus: przejście do stand. pliku wejściowego

seeing(Nazwa_Pliku) — uzgodnienie argumentu z nazwą bieżącego pliku wejściowego

tell(Nazwa_Pliku) — otwarcie wskazanego pliku, uczynienie go bieżącym plikiem wyjściowym

told — zamknięcie bieżącego pliku wyjściowego
MIM Prolog: powrót do poprzedniego (nic nie robi, gdy brak poprzednika); SICStus: przejście do stand. pliku wyjściowego

telling(Nazwa_Pliku) — uzgodnienie argumentu z nazwą bieżącego pliku wyjściowego

`0'`znak — liczba całkowita równa kodowi ASCII znaku (SICStus)

Operacje wejścia-wyjścia na termach

read(Term) — wczytanie z bieżącego pliku wejściowego termu zakończonego kropką; w przypadku końca pliku przekazanie stałej `end_of_file`

write(Term) — wypisanie termu na bieżący plik wyjściowy, zgodnie z obowiązującymi deklaracjami funktorów

writeq(Term) — analogicznie jak write; nazwy stałych, funktorów umieszczane w apostrofach (gdzie konieczne)

display(Term) — wypisanie termu na bieżący plik wyjściowy, w postaci standardowej (SICStus: na plik standardowy)

displayq(Term) — wypisanie termu na bieżący plik wyjściowy, w postaci standardowej, nazwy w apostrofach (MIM Prolog)

Przykłady

```
:- write( 'A' + b ), nl, write( '+'('A', b) ).
A + b
A + b
```

```
:- writeq( 'A' + b ).
'A' + b
```

```
:- display( 'A' + b ).
'+'( A, b )
```

```
:- displayq( 'A' + b ).
'+'( 'A', b )
```

Porównywanie termów

X = Y — uzgodnienie termów X i Y

X \= Y — X nie uzgadnia się z Y (MIM Prolog)

X == Y — identyczność termów X i Y

np. :- X == Y. – porażka
:- X = Y, X == Y. – sukces
:- 1 + 3 == 3 + 1. – porażka

X \== Y — termy X i Y nie są identyczne

uporządkowanie termów:

Term1 @ < Term2
Term1 @ =< Term2
Term1 @ > Term2
Term1 @ >= Term2

Przykład (standardowy sposób przetwarzania plików)

```
process_file( F ) :-
    seeing( Old ),      % (SICStus)
    see( F ),           % open F
    repeat,
       read( T ),
       process_term( T),
       T == end_of_file,
    !,
    seen,               % close F
    see( Old ).         % (SICStus)
```

Badanie właściwości termów

var(Term) — Term jest zmienną nieukonkretnioną

nonvar(Term) — Term nie jest zmienną nieukonkretnioną

integer(Term) — Term jest liczbą całkowitą

atom(Term) — Term jest stałą i nie liczbą całkowitą

atomic(Term) — Term jest atomem lub liczbą całkowitą

name(Atom, ListaZnaków) — Atom = zmienna lub stała, nazwa Atomu składa się ze znaków ListyZnaków

np. `?-` name(pr, L).
L = [112, 114]
`?-` name(const, "const").
`?-` name(alpha, Lista), name(X, [~ c | Lista]).
Lista = [97, 108, 112, 104, 97]
X = calpha

Wczytywanie programów

consult(ListaNazwPlików) — wczytanie i przetworzenie wszystkich podanych plików (teksty programów w Prologu)

postać uproszczona: sama lista nazw plików

np. `| :- [ 'plik1.pl', plik2 ].`

reconsult(ListaNazwPlików) — wczytanie i przetworzenie wszystkich podanych plików w trybie wymiany procedur

uwaga: jeden plik — wystarczy podać jego nazwę

Korzystanie z Prologu

dwa tryby pracy:

- – tryb definiowania procedur
- – tryb wykonywania dyrektyw i zapytań

MIM Prolog:

— początkowo tryb definiowania (zachęta: `|`)
— przejście do trybu zapytań: **mode**/0 (zmiana trybu)

SICStus Prolog:

— początkowo tryb zapytań (zachęta: `| ?-`)
— przejście do trybu definiowania:

```
| ?- [user].
| p(a).
| p(b).
```

— powrót do trybu zapytań:

```
| Control+D
| {user consulted, ... msec ... bytes}
yes
| ?- p(X).
X = a ?  ;
X = b ?
| ?-
```

zakończenie pracy:

— **halt/0**
— Control+D (SICStus), Control+Z (MIM Prolog)

Negacja w Prologu — piszemy `\+` zamiast $\neg$

Deklaracja operatorów (SICStus Prolog)

dyrektywa:

```
:- op(pierwszeństwo, typ, nazwa).
```

`pierwszeństwo` — liczba z zakresu 1..1200
`typ` — `xf, yf, yfx, xfx, xfy, fy, fx`
`nazwa` — nazwa (ew. w apostrofach) lub lista nazw

Typy operatorów:

pozycja	nawiasowanie		
	lewe	żadne	prawe
postfiksowe	yf	xf	—
infiksowe	yfx	xfx	xfy
prefiksowe	—	fx	fy

Przykłady standardowych operatorów:

```
:- op( 1200, xfx, [:-, -->] ).
:- op( 1200, fx, [:-, ?- ] ).
:- op( 500, yfx, [+, -] ).
:- op( 200, fy, [+, - ] ).
```

Korzystanie ze standardowych bibliotek

```
:- use_module(Plik).
:- use_module(Plik, ListaPredykatów).
```

Np. `:- use_module(library(lists)).`
załadowanie biblioteki operacji na listach