Vorbemerkungen

Allgemeines:

"Prolog is different, but not that different."

Datentypen:

Pascal	Prolog
integer	integer
real	float
char	ASCII-Codes
string	Listen von ASCII-Codes
file	Stream
Aufzählungs-Typen	Menge von Atomen
Variante Records	zusammengesetzte Terme
Arrays	Listen
	Mengen von Fakten
	Argumente eines Funktors
Pointer	Listen, Bäume einfach
	(sonst Atome verwenden)
—	Partielle Datenstrukturen
	(Terme mit Variablen)

Variablen (1)

Zuweisungen:

- Einer Prolog-Variablen kann nur ein einziges Mal ein Wert zugewiesen werden.

  Danach wird sie überall automatisch durch diesen Wert ersetzt und hört auf, eine Variable zu sein.

- Man braucht für jeden Wert eine neue Variable.

  Schleifen werden in Rekursionen übersetzt. Daher kann man bei jedem Schleifendurchlauf neue Variablen verwenden (s.u.).

Beispiel:

- **procedure** p(n: **integer**; **var** m: **integer**);
  begin
  m := n * 2;
  m := m + 5;
  m := m * m
  end
- p(N, M) :-
  M1 **is** N * 2,
  M2 **is** M1 + 5,
  M **is** M2 * M2.

Variablen (2)

Ein-/Ausgabeparameter:

- Aufspalten in ein Argument für den Eingabewert und eines für den Ausgabewert. „accumulator pair“

Beispiel:

- **procedure** double(**var** n: **integer**);
  begin n := n + n **end**
- double(N_In, N_Out) :-
  N_Out **is** N_In +N_In.

Globale Variablen:

- Den Prozeduren als Argumente mitgeben.
  Werte, die überall unverändert weitergereicht werden, heißen „context arguments“. Falls es zu viele sind, kann man sie in eine Struktur mit entsprechenden Zugriffsfunktionen verpacken.
- Notfalls mit `assert/retract` simulieren.
- Sepia-Prolog hat globale Variablen/Arrays, siehe `help(setval/2)`.

Beispiel:

- x := x + 1;
- x(X), retract(x(X)),!, X1 **is** X + 1, assert(x(X1)).

Fallunterscheidungen (1)

Bedingte Anweisungen:

- Mehrere Klauseln, der Rumpf enthält jeweils die vollständige Bedingung für die Anwendbarkeit.
- Mehrere Klauseln, jede beginnt mit der „else if“-Bedingung und dann einem Cut.
- (Bedingung `->` Then-Teil`;` Else-Teil)

Beispiel:

- **procedure** min(i, j: **integer**; **var** m: **integer**);
  begin if i < j **then** m := i **else** m := j **end**
- min(I, J, M) :- I < J, M = I.
  min(I, J, M) :- not(I < J), M = J.
  Natürlich wäre „min(I, J, I) :- I < J“ eleganter.
- min(I, J, M) :- I < J, !, M = I.
  min(I, J, M) :- M = J.
  „min(I,J,I) :- I<J, !.“ wäre falsch, z.B. „?- min(1,2,2).“
  Regel: Der Cut muß an genau die Stelle, bei der die richtige Klausel feststeht, nicht früher und nicht später.
- min(I, J, M) :- (I < J `->` M = I`;` M = J).

Fallunterscheidungen (2)

Bedingte Anweisungen (Forts.):

- Falls möglich, ist die beste Lösung, die Bedingung schon im Regelkopf auszudrücken.

  Größere Prologsysteme haben üblicherweise einen Index (Hashtabelle) über dem äußersten Funktor des ersten Arguments.

Beispiel:

- **procedure** p(c: color, **var** i: **integer**);
  begin
  case c **of**
  red: i := 1;
  green: i := 2;
  blue: i := 3;
  end
  end
- p(red, I) :- I = 1.
  p(green, I) :- I = 2.
  p(blue, I) :- I = 3.

  Eleganter wäre „p(red, 1).“ und entsprechend für green/blue.

Schleifen (1)

Allgemeines:

- Normalerweise als End-Rekursionen formuliert.

  Das Prolog-System gibt den Speicherplatz für eine Klausel-Aktivierung frei, wenn das letzte Literal aufgerufen wird und keine Alternativen mehr übrig sind (ggf. per Cut entfernen).

Beispiel:

- **procedure** length(l: list, **var** n: **integer**);

  n := 0;
  while l <> **nil do**
  n := n + 1; l := l^.next
  od

- length(L, N) :- length(L, 0, N).
  length(L, N_In, N_Out) :-
  L = [], !, N_Out = N_In.
  length(L, N_In, N_Out) :-
  N_Next **is** N_In + 1,
  [_|L_Next] = L,
  length(L_Next, N_Next, N_Out).

  Besser ist "length([], ...)" und "length([_|L_Next], ...)".

Schleifen (2)

Alternative:

- `repeat` und Backtracking.

  Beim Backtracking wird der belegte Speicherplatz vollständig freigegeben (nicht nur die Aktivierungsrecords der Klauseln, sondern auch konstruierte Terme).

Beispiel:

- **procedure** skip(c: **charakter**);
  var c1: **character**;
  begin
  repeat
  read(c1)
  until c1 = c
  end
- skip(C) :-
  repeat,
  get0(C1),
  C1 = C,
  !.

  Der Cut ist wichtig, wenn später im Programm Backtracking einsetzt. Die Schleife soll ja nicht noch einmal ausgeführt werden.

Schleifen (3)

For-Schleifen:

- Wieder mit Endrekursion.

Beispiel:

- **procedure** squares(n: **integer**);
  var i: **integer**;
  begin
  for i := 1 **to** n
  writeln(i * i)
  end
- squares(N) :-
  squares(1, N).
  squares(I, N) :- I > N, !.
  squares(I, N) :-
  I_Square **is** I * I,
  write(I_Square),
  nl,
  Next_I **is** I + 1,
  squares(Next_I, N).