Programmierstil

Allgemeines:

- Nichtlogische Konstrukte sparsam verwenden.

  Also cut, assert/retract und repeat/fail möglichst vermeiden. Programm in rein deklarativen Teil und Steuerprogramm einteilen.

- Bezeichner sollten aussagekräftig sein und nach einem einheitlichen Schema gewählt werden.

  z.B. enden in dem Buch von Sterling/Shapiro alle Listen auf „s“.

- Prädikate sollten max. 5-7 Argumente haben.

  Bei Bedarf mehrere Argumente in eine Struktur verpacken.

- Eingabeargumente vor Ausgabeargumenten.

  Natürlich ist bei Prolog nicht immer klar, was Eingaben und was Ausgaben sind. Falls es aber „Schlüsselattribute“ gibt, die also die anderen Argumente eindeutig bestimmen, so sollten diese zuerst kommen. Allgemeiner sollten Argumente, über denen eine Fallunterscheidung durchgeführt wird, nach vorne.

  Auch sonst sollte die Reihenfolge der Argumente einheitlichen Regeln folgen. Etwa kann man entsprechend der Datentypen eine Ordnung festlegen. Akkumulator-Paare sollten immer entsprechend der zeitlichen Reihenfolge nebeneinander stehen.

- Variablen, die nur einmal vorkommen, sollten mit einem Unterstrich beginnen.

- DB-Prädikate als „dynamic“ deklarieren.

  Compiler verlangen das. Aber auch bei Interpretern nützlicher Kommentar. Dort „dynamic“ selber deklarieren mit „dynamic(_).“.

Programm-Layout („Pretty Printing“)

Vorschläge:

- Alle Klauseln zu einem Prädikat sollten hintereinander in einer Datei stehen.

  Die meisten Prolog-Compiler erlauben gar keine andere Anordnung.

- Jedes Rumpfliteral sollte eingerückt auf einer eigenen Zeile stehen.

- Disjunktionen sollten deutlich erkennbar sein.

  Es wird folgendes Einrückungsschema vorgeschlagen:

```
(     p(X, a),
      q(f(X,a))
;     r(X, b)
      s([b|X], _)
)
```

- Keine Zeile sollte länger als 79 Zeichen sein.

- Bei Prädikaten sollte zwischen den Argumenten ein Leerzeichen stehen, aber nicht bei Funktoren.

- Die Prädikatdefinitionen sollten „top-down“ angeordnet werden.

  d.h. die aufgerufenen Hilfsprädikate nach den aufrufenden Prädikaten.

Kommentierung

Empfehlungen:

- Die Bedeutung jedes Prädikates (d.h. die Relation) sollte deklarativ beschrieben werden.

  z.B. `member(X,Y)` $\Longleftrightarrow$ `X` ist Element der Liste `Y`.

- Typ und Bindungsart (Mode) jedes Arguments sollten dokumentiert werden.

  Obwohl Prolog keine Typdeklarationen verlangt, können viele Prädikate nicht beliebige Terme als Eingaben verarbeiten.

  Mode-Deklartionen besagen, welche Argumente der Prädikate bei einem Aufruf gebunden sind und welche nicht. Üblich sind die Zeichen `+` für gebundene Variablen, `-` für ungebundene Variablen und `?` für beliebige Argumente. Bei Sepia-Prolog bedeutet `+` nur, daß es sich nicht um eine Variable handelt, `++` garantiert, daß das Argument ein variablenfreier Term ist.

  Bei Sepia-Prolog kann man Modes in der Form `:- mode fib(+,?)` deklarieren. Dadurch wird etwas effizienterer Code erzeugt. Der Compiler überprüft diese Angaben nicht. Wenn das Prädikat mit falschen Argumenten aufgerufen wird, werden falsche Antworten geliefert.

- Seiteneffekte sollten hervorgehoben werden.

- Einzelne Klauseln und Literale benötigen nur in Ausnahmefällen einen Kommentar.

  Besser Kommentar in größeren Blöcken konzentrieren.

- Jede Datei sollte einen Kommentar-Kopf haben.

  Autor, Datum, Programmiersprache/Version, Beschreibung, ...

Effizienz (1)

Optimierungs-Möglichkeiten:

- Man nütze eventuell vorhandene Indexe.

  Viele bessere Prologs haben eine Hashtabelle, die zu dem äußersten Funktor des ersten Arguments die zugehörige Klausel liefert.

- Deterministische Aufrufe sollten für das Prolog-System auch erkennbar sein.

  Ein Prädikat heißt deterministisch für einen bestimmten Mode (Bindungsmuster), wenn es höchstens eine Lösung hat.

  Das Prolog-System muß hierzu erkennen können, daß höchstens eine der Klauseln anwendbar ist (und die aufgerufenen Prädikate ebenfalls deterministisch sind). Hier ist der Index über dem ersten Argument und ggf. ein Cut nützlich. Nur sehr wenige Prologs erkennen, daß sich bestimmte eingebaute Prädikate gegenseitig ausschließen.

  Ein einfacher Test ist, das Prädikat entsprechend aufzurufen. Fragt das System nach der Ausgabe einer Lösung, ob man noch weitere Lösungen wünscht, so hat es nicht erkannt, daß das Prädikat deterministisch ist.

  Aber: Nicht zu viele Cuts! Z.B. nur einen Cut pro äußere Schleife.

- Man nütze die Endrekursions-Optimierung.

  Der rekursive Aufruf muß das letzte Rumpfliteral sein und es muß für den Interpreter/Compiler erkennbar sein, daß keine weiteren Alternativen für das aktuelle Ziel mehr bestehen. Am einfachsten zu erreichen durch einen Cut direkt vor dem rekursiven Aufruf.

  Besser ist es natürlich, den Index über dem ersten Argument auszunutzen. Bei der letzten Klausel ist ebenfalls kein Cut nötig. In beiden Fällen müssen die aufgerufenen Prädikate deterministisch sein.

Effizienz (2)

Optimierungs-Möglichkeiten, Forts.:

- Im Regelrumpf sollte man die einfachen Tests nach vorne ziehen („fail as early as you can“).

  Wenn sich herausstellt, daß die Klausel nicht anwendbar ist, sollte man so wenig wie möglich überflüssige Arbeit investiert haben.

- Man sollte möglichst sparsam mit dem Aufbauen neuer Strukturen sein.

  D.h. mit dem Speicherplatz auf dem Heap.

- Mit „fail“ erzwungenes Backtracking kann Speicherplatz freigeben.

  Alle auf dem Heap angelegten Strukturen sind nicht mehr zugreifbar und werden freigegeben.

- „Assert“ und „retract“ sind sehr aufwendig.

- Listen werden meist speziell behandelt.

  Obwohl man sich Listen mit einem beliebigen zweistelligen Funktor aufbauen kann, ist es also günstiger die Standard-Darstellung von Listen zu verwenden.

- Variablen, die nur im Kopf und im ersten Rumpfliteral vorkommen, werden speziell behandelt.

  Solche „temporären Variablen“ brauchen keinen Platz auf dem Stack. Es ist besonders günstig, wenn sich die Argument-Position dieser Variablen nicht ändert.

Modularisierung

Nachtrag zu eingebauten Prädikaten:

- Bessere Prolog-Systeme erlauben es, die Sichtbarkeit der Prädikate einzuschränken.

  Immer möglich: Programm auf mehrere Quelldateien verteilen.

- `:- module(mod).`

  Alle folgenden Klauseln gehören zum Modul `mod`.

- `:- export p/1, q/2.`

  Die Prädikate `p/1` und `q/2` können von einem anderen Modul importiert werden.

- `:- import p/1, q/2 from mod.`

  Die Prädikate `p/1` und `q/2` sind im Modul `mod` definiert.

- `:- global p/1, q/2.`

  Wie `export`, aber es ist kein `import` nötig.

- Alle diese Aufrufe sind Sepia-Prolog spezifisch.

  Bei Quintus-Prolog werden Modul- und Export-Deklaration verbunden zu „`:- module(mod, [p/1, q/2])`“. Das Importieren geschieht dort mit „`:- use_module(mod, [p/1, q/2])`“.

  Bei Turbo-Prolog deklariert man globale Typen (`global domains`) und Prädikate (`global predicates`) in einer eigenen Datei und liest diese dann mit `include "datei.pro"` in jede Quelldatei ein.

- Schwierig: Prädikate als Argumente an Prädikate in anderen Modulen übergeben.

  Lösung bei Quintus-Prolog z.B.: „`:- meta_predicate not(:).`“