Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Informatik und Prakt. Mathematik
Prof. Dr. M. Hanus, Dr. Frank Huch

24118 Kiel
Christian-Albrechts-Platz 4
Tel.: 0431 / 880-7277

10. Übung „Prinzipien von Programmiersprachen“

Bearbeitung bis zum 25. Januar 2004

Aufgabe 1

Überprüfen Sie mit dem Inferenzsystem zur Berechnung des allgemeinsten Unifikators (*mgu*) die folgenden Termpaare auf Unifizierbarkeit:

a) `f(g(a,X),h(Y,c(Z)),h(X,V))` und `f(Y,h(g(Z,c(V)),c(a)),h(c(V),c(c(Z))))`

b) `h(f(V,Y),h(Z,f(a,X)))` und `h(f(h(X,g(Y)),a),h(Z,g(a)))`

c) `f(A,g(C,B))` und `f(g(B,D),g(v(t,B),h(D)))`

d) `f(A,B,C)` und `f(g(X,X),g(A,A),g(B,B))`

Geben Sie die allgemeinsten Unifikatoren an, bzw. überlegen Sie sich, warum die Unifikation fehlgeschlagen ist.

Aufgabe 3

Gegeben sei das Prolog-Programm:

```
p(a,b).
p(b,a) :- q(a,b).
q(Y,X) :- p(X,Y).
```

und die Auswahlregel „wähle das letzte Literal“. Geben Sie die SLD-Ableitungen einschließlich Resolventen und Unifikatoren eines jeden einzelnen Schritts für die folgenden Anfragen an:

a) `?- p(a,b),q(b,a).` b) `?- q(a,b).` c) `?- q(b,b).`

Welche der Ableitungen ist eine erfolgreiche SLD-Ableitung?

Aufgabe 3

Definieren Sie folgende Listenoperationen durch Prolog-Klauseln:

a) Umkehrung der Reihenfolge aller Elemente einer Liste.
(Prädikat `reverse(List,RevList)`)

b) Überprüfung, ob eine Liste Teilliste einer anderen Liste ist. Dabei ist eine Teilliste von $[E_1,\ldots,E_n]$ ($n \geq 0$) entweder die leere Liste oder eine Liste der Form $[E_i,\ldots,E_j]$ ($1 \leq i \leq j \leq n$). (Prädikat `subList(SubList, List)`)

c) Definieren Sie ein Prädikat `perm`, welches für zwei Listen prüft, ob sie Permutationen von einander sind. Falls das zweite Argument eine logische Variable ist, soll diese nichtdeterministisch an alle möglichen Permutationen gebunden werden. (Idee: gehen Sie ähnlich wie bei der Haskell-Implementierung vor, allerdings verwenden Sie Nichtdeterminismus anstelle der Listen von Listen).

d) Mit Hilfe der Programmiertechnik "generate-and-test", kann man Lösungen für Suchprobleme finden. Hier wollen wir Lösungen für das Haus vom Nikolaus suchen. Die Eckpunkte des Hauses seien durch die Zahlen 1 bis 5 repräsentiert. Eine Kante im Haus vom Nikolaus, sei dann durch ein Paar zweier Zahlen definiert. Alle Kanten des Hauses seien durch folgendes Prädikat gegeben:

```
houseLines([(1,2),(1,3),(2,3),(2,4),(2,5),(3,4),(4,5),(4,5)]).
```

Hierbei ist zu beachten, dass jede Kante nur genau einmal vorkommt. In einem gültigen Nikolauspfads können die Kante aber auch in der umgekehrten Richtung verwendet werden. Zum Finden einer Möglichkeit, das Haus vom Nikolaus zu malen, kann dann folgendes Prädikat verwendet werden:

```
findNiko(Niko) :- houseLines(Lines),
                  perm(Lines,NikoUnDir),
                  flip(NikoUnDir,Niko),
                  niko(Niko).
```

Zunächst wird die Variable `NikoUnDir` an alle möglichen Permutationen der Hauskanten gebunden. Diese werden dann noch nicht-deterministisch geflippt und schließlich wird überprüft, ob es sich tatsächlich um einen gültigen Linienzug (ohne Unterbrechung) handelt. Diese Suche ist zwar nicht sehr effizient, dafür aber sehr einfach und liefert mit etwas Geduld (diese Zeit hat man aber sicher durch die kurze Entwicklungszeit gespart) die Lösungen.

Fügen Sie Code für die Prädikate `flip` und `niko` hinzu.

Nun können Sie mit folgender Anfrage die Anzahl aller Möglichkeiten das Haus vom Nikolaus zu malen bestimmen (Bindung von `Sols`):

```
findall(X,findNiko(X),Ys),length(Ys,Sols).
```

e) Wer Lust hat, kann noch über Optimierungen nachdenken oder eine Lösung in Haskell versuchen.

Hinweis zur Benutzung von Prolog:
Das auf den Instituts-Rechnern installierte Sicstus-Prolog-System lässt sich mit

```
/home/prolog/bin/sicstus
```

starten. Prolog-Programme sollten die Endung `.pl` haben. Diese Programme lassen sich mit

| ?- [$file_1$,...,$file_n$].

in das Prolog-System laden.
Weitere Informationen, auch zu frei verfügbaren Prolog-Systemen, finden Sie auf der Web-Seite zur Vorlesung.