Christian-Albrechts-Universität zu Kiel
AG Programmiersprachen und Übersetzerkonstruktion
Prof. Dr. M. Hanus

Christian-Albrechts-Platz 4
24118 Kiel
Tel.: 0431 / 880-7271

10. Übung zur Vorlesung „Prinzipien von Programmiersprachen“

Wintersemester 2008/2009

Abgabe: 3. Februar 2009 in der Vorlesung

Aufgabe 37

a) Definieren Sie in Prolog 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).

b) 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),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(4,5)]).
```

Hierbei ist zu beachten, dass jede Kante nur genau einmal vorkommt. In einem gültigen Nikolauspfad können die Kanten 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 `NumSols`):

```
?- findall(X,findNiko(X),Ys),length(Ys,NumSols).
```

Aufgabe 38

In dieser Aufgabe sollen Sie ein Prolog-Programm zum Lösen von Sudoku schreiben. Ein Sudoku besteht aus 9×9 Feldern, die zusätzlich in 3×3 Blöcke mit jeweils 3×3 Feldern aufgeteilt sind. In einem gelösten Sudoku enthält jede Zeile, jede Spalte und jeder Block alle Zahlen von 1 bis 9 jeweils genau einmal. In einigen der Felder sind bereits Zahlen vorgegeben. Hier ist ein Beispiel für ein Sudoku-Problem:

9			2			5		
	4			6			3	
		3						6
			9			2		
				5			8	
		7			4			3
7						1		
	5			2			4	
		1			6			9

Ein einfaches „generate-and-test“-Verfahren wie in Aufgabe 37 führt zu keiner akzeptablen Lösung. Überlegen Sie sich dazu, wie viele Möglichkeiten dazu für das obige Sudoku-Problem durchprobiert werden müssten.

Zu einer praktikablen Lösung kommen Sie dagegen, wenn Sie Constraints über endlichen Bereichen einsetzen, wie dies in der Vorlesung anhand des CLP(FD)-Programms zur Lösung des `send+more=money` Problems demonstriert wurde. Erstellen Sie daher auf ähnliche Weise, insbesondere durch Ausnutzung von `all_different`-Constraints, ein CLP(FD)-Programm zur Lösung von Sudoku-Problemen.