Lehrstuhl für Softwaretechnik und Programmiersprachen
Professor Dr. Michael Leuschel

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Programmiersprachen II – Wintersemester 2011/2012
Wiederholung zu Prolog
Besprechung der Aufgaben in der nächsten Übung.

1 Einführung

Die Aufgaben hier dienen dazu, um grundlegende Prolog-Kenntnisse aufzufrischen und evtl. Fragen/Probleme in den Übungen klären zu können. Die Lösungen zu den Aufgabe sind in der Regel recht kurz und einfach – vorausgesetzt man hat die zu Grunde liegenden Mechanismen verstanden.

Um den Umgang mit Prolog zu üben, empfehlen sich auch die Übungsblätter 1–8 der Vorlesung „Softwaretechnik und Programmiersprachen“ aus dem Sommersemester 2011

Das Bearbeiten der Aufgaben ist für die Vorlesung Programmiersprachen 2 ***nicht*** verpflichtend. Eine freiwillige Abgabe ist möglich unter: John.Witulski@uni-duesseldorf.de. Termin für die Besprechung ist Donnerstag der 15.12.2011 14:30 in 25.12.02.55.

2 Aufgaben

Aufgabe 1 (Rekursion und Listenbearbeitung)

a) Schreiben Sie ein Prädikat `remove/3`, das als erstes Argument eine Liste als Eingabe erhält und als zweites Argument einen beliebigen Term. Im dritten Argument wird dann eine Liste geliefert, die dem ersten Argument gleicht, außer dass alle Vorkommen des zweiten Terms entfernt wurden. Zum Beispiel:

```
?- remove([c,b,x,d,x,z],x,L).
L = [c,b,d,z] ?
yes
```

Überprüfen Sie, ob Ihre Implementation mehr als eine Lösung liefert (was nicht korrekt wäre).

Tipp: Für eine Implementation bieten sich die Konstrukte `\=` (nicht unifizierbar) oder `(if -> then; else)` an.

b) In einer Datenstruktur sollen Name/Wert-Paare abgelegt werden (eine „Map“). Schreiben Sie dazu drei Prädikate `empty/1`, `insert/4` und `lookup/3`. Dabei soll `empty/1` eine leere Datenstruktur im Argument liefern. `insert/4` fügt der Map im ersten Argument einen neues Name/Wert-Paar (2. und 3. Argument) hinzu und liefert die so entstandene neue Map im 4. Argument zurück. `lookup/3` bekommt als erstes Argument eine Map, als zweites einen Namen und liefert im dritten Argument den zugehörigen Wert.

Zum Beispiel wird hier hintereinander eine leere Datenstruktur (`_M1`) erzeugt, zwei Elemente hinzugefügt ($a \to 5$, $b \to 7$) und dann eines ausgelesen:

```
?- empty(_M1), insert(_M1,a,5,_M2), insert(_M2,b,7,_M3), lookup(_M3,a,Value).
Value = 5 ?
yes
```

Tipp: Benutzen sie eine Liste, in der jedes Element ein Name/Wert-Paar repräsentiert.

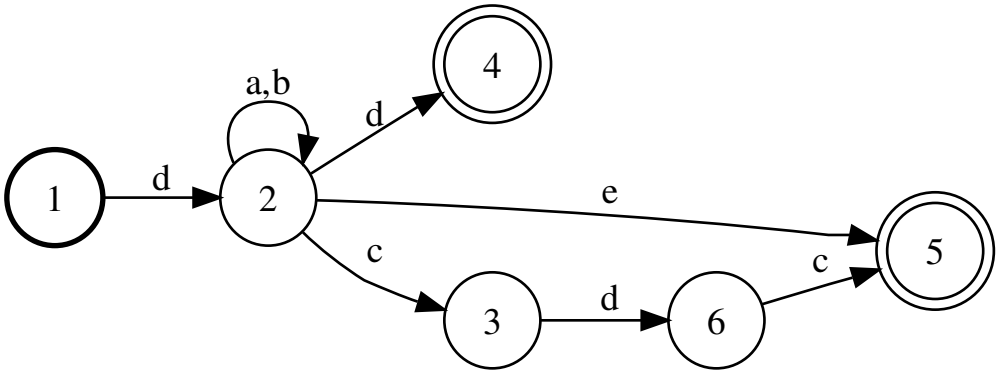

Abbildung 1: Automat, der dick umrandete Zustand 1 ist der Startzustand

Aufgabe 2 (Automaten)
Überlegen Sie sich, wie man einen Automat wie in Abbildung 1 als Faktenbasis speichert. Schreiben sie dazu ein Prädikat `accept/1`, das eine Liste übergeben bekommt und genau dann wahr ist, wenn der Automat das übergebene Wort akzeptiert. Zum Beispiel:

```
?- accept([d,a,b,a,b,b,b,c,d,c]).
yes
?- accept([d,a,b,a,e,d,c]).
no
```

Aufgabe 3 (Arithmetik)
a) Warum liefert die Anfrage

```
?- 6*3 = 18.
```

die Antwort `no`? Wie müsste die Anfrage formuliert werden, damit korrekterweise mit `yes` geantwortet wird?

b) Schreiben sie ein Prädikat `llength/2`, das als erstes Argument eine Liste erhält und als zweites Element die Länge der Liste zurück gibt. Zum Beispiel:

```
?- llength([a,b,c],L).
L = 3 ?
yes
```

c) Für eine Taschenrechner-Anwendung liefert ein Parser die Eingabe des Benutzers als Syntaxbaum mit Zahlen als Blättern und Verknüpfungen wie Plus/Minus/Mal/Geteilt als innere Knoten. Der Syntaxbaum ist als Prolog-Term mit den Elementen `plus/2`, `minus/2`, `times/2` und `div/2` und `num/1` beschrieben.

Zum Beispiel hat der Ausdruck $5 * 3 + (7 - 4)$ den Syntaxbaum in Abbildung 2. Der zugehörige Prolog-Term wäre `plus( times(num(5),num(3)), minus(num(7),num(4)) )`.

Schreiben Sie ein Prädikat `calc/2`, das als erstes Argument einen Ausdruck wie oben bekommt und im zweiten Argument das Ergebnis liefert. Zum Beispiel:

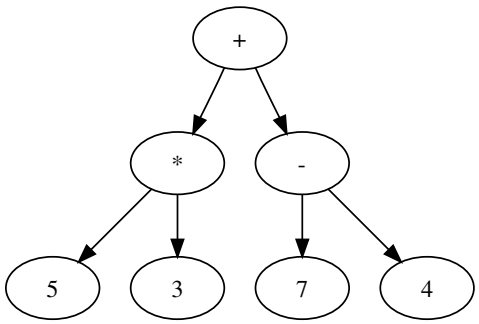

Abbildung 2: Syntaxbaum des Ausdrucks $5 * 3 + (7 - 4)$

```
?- calc( plus(times(num(5),num(3)),minus(num(7),num(4))), N).
N = 18 ?
yes
```

Aufgabe 4 (SAT-Solver in Prolog)
Eine aussagenlogische Formel sei als Prolog-Term gegeben. Neben Konstanten und Variablen können Formeln mittels Und, Oder und Negation verknüpft werden. Für die Konstanten *true* und $false$ werden die Terme `cst(true)` bzw. `cst(false)` und für die Verknüpfungen `and/2`, `or/2` und `not/1` verwendet. Die aussagenlogische Formel

$$\neg(true \wedge false) \vee false$$

sähe als Prolog-Term also folgendermaßen aus:

```
or( not( and( cst(true), cst(false) ) ), cst(false) )
```

Schreiben Sie ein Prädikat `istrue/1`, dass genau dann wahr ist, wenn die übergebene Formel wahr ist. Zum Beispiel:

```
?- istrue( or(not(and(cst(true), cst(false))), cst(false)) ).
yes
?- istrue( or(not(cst(true)), cst(false)) ).
no
```

Falls in der Formel Variablen verwendet werden, sollen mögliche Lösungen gefunden werden:

```
?- istrue( or(not(and(cst(true), cst(A))), cst(B))).
A = false ? ;
B = true ? ;
no
```

Tipp: Verwenden sie für die Negation nicht die Prolog-Negation, sondern implementieren Sie ein zweites Prädikat `isfalse/1`, das angibt, ob eine Formel falsch ist.