

**Programmiersprachen II – Wintersemester 2011/2012**  
**Übungsblatt 2**  
**Besprechung der Aufgaben in der nächsten Übung am 19.01.2012.**

## 1 Abgabe

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 19.01.2012 14:30 in 25.12.02.55.

## 2 Aufgaben

### Aufgabe 1 (Listen und Tupel)

Listen sind in Prolog durch den Punktoperator `'.'`/2 gegeben. Dessen erster Parameter ist ein beliebiges Element. Der zweite Parameter ist entweder ein weiterer Punktoperator mit zwei Parametern oder aber die leere Liste `[]`. Die leere Liste wird durch `[]` dargestellt. Bitte geben Sie die folgenden Abfragen in Ihren Prolog Interpreter ein und versuchen Sie diese zu verstehen:

```
?- [1,2] = '.'(1,'.'(2,[])).  
yes
```

```
?- [] = '.'(A,B).  
no
```

```
?- [a] = '.'(A,B).  
A = a,  
B = [] ?  
yes
```

```
?- [a,b,c,d] = [H|T].  
H = a,  
T = [b,c,d] ?  
yes
```

```
?- [a,b,c,d] = '.'(H,T).  
H = a,  
T = [b,c,d] ?  
yes
```

```
?- X = [1,2,3], X =.. ['.'|R].  
R = [1,[2,3]],  
X = [1,2,3] ?  
yes
```

Benutzen Sie dieses Wissen als Grundlage für diese Aufgabe.

a) Schreiben Sie erneut das Prädikat `list/1`, welches prüft, ob der gegebene Parameter eine Liste ist.

Benutzen Sie hierfür aber diesmal die Punktnotation.

Beispiel:

```
?- list([a,b,c]).  
yes
```

- b) Schreiben Sie ein Prädikat `listify/2`, welches als ersten Parameter ein Tupel erhält. Als zweiten Parameter soll man eine Liste mit den Elementen des Tupels zurückerhalten. Benutzen Sie für dieses Prädikat die Punktnotation.

Beispiel:

```
?- listify((a,b,c,d), Liste).  
Liste = [a,b,c,d] ?  
yes
```

**Tipp:** Wenn Sie Schwierigkeiten haben, sehen Sie sich diese Abfrage in Prolog an:

```
?- (a,b,c,d) = (a,(b,(c,d))).  
yes
```

### Aufgabe 2 (Interpreter einer imperativen Programmiersprache)

Sehen Sie sich das Kapitel 7.1 im Skript an. Nutzen Sie für diese Aufgabe einen der vorgefertigten Interpreter, die Sie auf der STUPS-Seite [http://www.stups.uni-duesseldorf.de/w/Programmiersprachen\\_2,\\_WiSe\\_11](http://www.stups.uni-duesseldorf.de/w/Programmiersprachen_2,_WiSe_11) finden können. Benutzen Sie entweder `6_imp_int.pl` oder `6_imp_int_check.pl`.

- a) Sehen Sie sich folgende Abfrage an:

```
?- eint( (def x; x:= 5; def z; x:= $x+1; z:= $x+($x+2)) , [],R).  
R = [z/14,x/6] ?
```

Bei der Suche nach weiteren Lösungen wird ein Fehler auftreten. Finden und beheben Sie diesen. Am einfachsten ist die Benutzung des `Cut(!)`<sup>1</sup>.

- b) Erweitern Sie den Interpreter um mehr Operatoren: Multiplikation (`*`) und Potenz (`**`), ausserdem noch die boolesche Operation: (`>`).
- c) Schreiben Sie den Interpreter so um, dass das Zeichen "\$" nicht mehr nötig ist. Vergewissern Sie sich, dass Sie ebenfalls nur eine Lösung erhalten.

Beispiel:

```
?- eint( (def x; x:= 5; def z; x:= x+1; z:= x+x+2) , [],R).  
R = [z/14,x/6] ?  
yes
```

- d) Erweitern Sie den Interpreter um eine Typisierung. Die Variablen sollen über das Schlüsselwort `int` bzw. `bool` statt über `def` deklariert werden. Durch die Typisierung können Variablen auch einen booleschen Wert erhalten. Schreiben Sie zur Hilfe ein Prädikat `lookup_type/3`, welches den Typ einer Variablen zurückgibt. Benutzen Sie die Typen `int` und `bool`.

Beispiel:

---

<sup>1</sup>Eine Einführung zum `Cut` finden Sie unter <http://www.amzi.com/AdventureInProlog/a13cut.php> und unter [http://www.sics.se/sicstus/docs/3.7.1/html/sicstus\\_5.html#SEC48](http://www.sics.se/sicstus/docs/3.7.1/html/sicstus_5.html#SEC48)

```
?- eint((bool x), [],_Out), lookup_type(x,_Out,Type).
Type = bool ?
yes
```

```
?- eint( (bool y; y:= 1 < 2;int z; if(y, z:=1, z:=2)), [],_R),lookup(z,_R,Z).
Z = 1 ?
yes
```

- e) Geben Sie eine Fehlermeldung aus, sobald ein Typfehler auftritt.  
Beispiel:

```
?- eint( (bool y; y:= 1), [],R).
type_error(y)
no
```

### Aufgabe 3 (Dynamische Fakten)

- a) Schreiben Sie einen Generator für reguläre Grammatiken `generate/1`. Als Parameter bekommt das Prädikat eine Liste mit Regeln der Form `trans/3` (=trans(von, Zeichen, nach)) übergeben. Nutzen Sie für Startzustände die Klausel `trans/2` (=trans(von, Zeichen)) und für Endzustände `trans/1` (=trans(nach)). Bauen Sie mit diesen Regeln einen Automaten und schreiben Sie ein Prädikat `accept/1`, welches eine gegebene Zeichenkette auf die angegebenen Regeln prüft (Siehe Blatt 1, Aufgabe 3). Speichern Sie die Regeln mit Hilfe dynamischer Fakten ab.

Beispiel:

```
?- generate([trans(1,c),trans(1,c,2),trans(2,b,3), trans(2), trans(3)]).
yes
```

```
?- generate([trans(1,a),trans(1,a,2),trans(2,b,3), trans(2,a,2),trans(3)]),accept([a,a,b]).
yes
```

```
?- generate([trans(1,a),trans(1,a,2),trans(2,b,3), trans(2,a,2),trans(3)]),accept([a,a,a]).
no
```

**Hinweis:** Achten Sie darauf, dass Sie vor dem Generieren des Automaten alle vorherigen dynamischen Fakten mit `retractall/1` löschen, sonst gibt es böse Überraschungen.

- b) Hin und wieder kommt es vor, dass diesselbe Zeichenkette mit verschiedenen Wegen durch die Regeln zum Endzustand kommen kann. Schreiben Sie das Prädikat `print_all_solutions/1`, welches als Parameter eine Zeichenkette bekommt. Es sollen alle Wege, wie diese Zeichenkette zum Endzustand kommen kann, ausgegeben werden. Benutzen Sie dafür die Klausel `way/1`.

Beispiel:

```
?- generate([trans(1,a),trans(1,a,2),trans(2,a,3), trans(2,a,2),trans(3),trans(2)]).
yes
?- print_all_solutions([a,a,a]).
way([1,2,2,3])
way([1,2,2,2])
yes
```

**Hinweis:** Da Sie hier mit dynamischen Fakten arbeiten, ist eine „fail-loop“ zur Ausgabe nützlich.

#### Aufgabe 4 (Brainfuck)

Wir betrachten folgendes Maschinenmodell:

- Der Speicher besteht aus einem nicht begrenzten Band von Speicherzellen.
- Es gibt einen Schreib-/Lesekopf, der von einem Programm gesteuert wird.
- Es können folgende acht Instruktionen ausgeführt werden:
  1. > : Bewegt den Schreib-/Lesekopf auf dem Band ein Feld nach Rechts.
  2. < : Bewegt den Schreib-/Lesekopf auf dem Band ein Feld nach Links.
  3. + : Erhöht den aktuellen Wert der Speicherzelle um eins.
  4. - : Erniedrigt den aktuellen Wert der Speicherzelle um eins.
  5. . : gibt den aktuellen Zellenwert als ASCII Zeichen auf der Standardausgabe aus.
  6. , : List ein Zeichen von der Standardeingabe und speichert den ASCII Wert des Zeichens in der aktuellen Zelle.
  7. [ : Springt vorwärts hinter den passenden ]-Befehl, wenn der aktuelle Zellenwert 0 ist.
  8. ] : Springt zurück zu passenden [-Befehl.
- Jedes andere Zeichen wird ignoriert.

Die so definierte Sprache heisst Brainfuck (siehe auch <http://de.wikipedia.org/wiki/Brainfuck>)

- a) Schreiben Sie einen Interpreter-Prädikat `brainfuck/3` für Brainfuck. Der erste Parameter ist die Standardeingabe, d.h. ein Eingabestring, auf den das Programm im Laufe der Berechnung zugreifen darf. Der zweite Parameter ist die Standardausgabe. Der letzte Parameter ist ein Brainfuck-Programm als String.

Ein Beispiel ist:

```
?- brainfuck("",R,"+++++++ [>+++++>+++++++>+++>+<<<<-]
>+.>+.+++++. .+++>+.<<+++++++>+. .+++ .----- .----- .>+.>.").
R = "Hello World!"
yes
```