Übungen 7: Rekursive Listenverarbeitung

Programmiertechniken in der Computerlinguistik I · Wintersemester 2001/2002

Um keine Zeit mit Tippen und Tippfehlerkorrektur zu verschwenden, findest du die Programmtexte dieser Übung unter http://www.ifi.unizh.ch/cl/siclemat/lehre/ws0102/pcl1/uebung7.txt

1. Das letzte Element

Definiere das Prädikat last/2, das als erstes Argument eine Liste nimmt und als zweites Argument das letzte Element dieser Liste zurückliefert.

```
?- last([kiwi,apfel,birne,zwetschge], Letztes).
Letztes = zwetschge
```

Falls die Liste kein Element enthält, soll das Prädikat scheitern:

```
?- last([], Letztes).
no
```

Das Prädikat soll auch als Generator funktionieren, das bei Backtracking alle möglichen Listen aufzählt, die als letztes Element das zweite Argument enthalten.

```
?- last(Liste, zwetschge).
Liste = [zwetschge] ;
Liste = [_A,zwetschge] ;
Liste = [_A,_B,zwetschge] ;
…
```

2. Papageien

a) Nimm die Abbildungsprädikat papagei/2 aus der Vorlesung. Teste das Programm mit der Anfrage:

```
?- papagei([du,bist,ein,papagei], Antwort).
```

Lasse dir alle Lösungen durch Eingabe des Strichpunkts ausgeben. Überlege, wie die Reihenfolge der Lösungen zustande kommt. Verändere das Programm so, dass es nur noch die erwartete Antwort gibt.

(Tip: Recycle dein Wissen aus "Daten- und Kontrollfluss")

b) Papageien-Zensur: Modifizere das papagei/2-Programm so, dass beleidigende Ausdrücke wie etwa "doof" oder "doofer" aus den Papageienantworten herausgefiltert werden. Beispielanfrage:

```
?- papagei([du,bist,doof], Antwort).
Antwort = [ich,bin] ;
no
```

3. Nomen zählen

Definiere ein Prädikat zaehle_nomen/2, das berechnet, wieviele Nomen in einer Wortliste vorkommen.

```
?- zaehle_nomen([the,boy,stood,on,the,deck], Nomen).
Nomen = 2 ;
no
?- zaehle_nomen([mhhm,sorry], Nomen).
Nomen = 0 ;
no
```

Nomen sollen als lexikalische Fakten wie "`n(boy)`" oder "`n(deck)`" definiert sein.

4. Rekursive Beweisbäume

a) Zeichne den Beweisbaum für die ersten 3 Lösungen der folgenden Anfrage

```
?- member(E, L).
L = [E|_A] ? ;
L = [_A,E|_B] ? ;
L = [_A,_B,E|_C] ?
yes
```

Verwende dabei folgende Definition ohne anonyme Variablen, um die Substitutionen explizit zu halten:

```
member(X, [X|Rest]).
member(X, [Y|Rest]) :-
  member(X, Rest).
```

b) Vergleiche die Variablenbindungen des Prolog-Interpreters mit den Bindungen, die in deinem Beweisbaum entstanden sind. – Die Bindungen an die Variablen der Anfrage liest du vom Beweisbaum ab, indem du von der Beweisbaumwurzel den Pfad zu einem leeren Ziel verfolgst und dabei immer die an den Kanten vermerkten Substitutionen auf die Variablen anwendest. Was für ein Problem gibt es?

c) Um dem Problem von (b) zu entgehen, hat Peter P. für sich die Beweisregel A aus der Folie 18 der Vorlesung "Beweisen" wie folgt ergänzt (Neues **fett**):

"Suche einen Regelkopf, der mit dem ersten Term des Ziels unifiziert, **wobei zuerst alle Variablen der Regel durch frische (d.h. noch nie verwendete) Variablennamen substituiert werden.**"

Wie sieht der Beweisbaum aus, der nach der Regel von Peter P. gebildet wurde? Ist das Problem (b) damit vollständig behoben?