Übungen 4: Occur Check, Debuggen, Arithmetik

Programmiertechniken in der Computerlinguistik I · Wintersemester 2000/2001

Um keine Zeit mit Tippen und Tippfehlerkorrektur zu verschwenden, findest du die Programmtexte dieser Übung unter http://www.ifi.unizh.ch/cl/siclemat/lehre/ws0001/pcl1/uebung4.txt

1. Verflixte Unifikation

Überlege dir, ob die Anfrage

```
?- f(X,g(Y)) = X, f(X,g(Y)) = Y.
```

bewiesen werden kann. Teste sie am Prolog-Interpreter und stelle den an X gebunden Term graphisch dar, falls die Anfrage gelingt.

2. Ei und Huhn-Trace

Wer war zuerst da, das Ei oder das Huhn? Schreibe ein Programm, das sich in Form der folgenden Regeln dem Problem widmet:

Ein Huhn gibt es, falls es ein Ei gibt
Ein Ei gibt es, falls es ein Huhn gibt.

a) Definiere die entsprechenden 2 Prädikate, schalte den Tracing-Modus ein und versuche die Existenz des Huhns aus dem Ei und umgekehrt zu beweisen.

b) Überlege dir, was die Erfahrungen aus (a) für das Ei-Huhn-Problem bedeuten.

c) Überlege dir, was die Erfahrungen aus (a) für Prädikatsdefinitionen und den Prolog-Beweiser bedeuten.

(Hinweis: Man kann den Beweis eines Prädikates beim Tracing mit Eingabe von 'a' wie abort abbrechen.

3. Count Down

Die beiden Hacker R. Ekur und S. Iv. haben ein Programm geschrieben, das einen Count Down ausgibt:

```
count_down(Zahl) :-
  Zahl < 0,
  write('Gooo!!!').
count_down(Zahl) :-
  Zahl >= 0,
  write(Zahl),
  nl,
  Zahl_Minus_1 is Zahl - 1,
  count_down(Zahl_Minus_1).
```

a) Teste ihr Prädikat mit positiven und negativen Zahlen als Argument.

b) Verwende *trace* oder *debug*, um zu verstehen, was da vor sich geht, und wieso das Programm funktioniert.

4. Korpus-Repräsentation und -Verarbeitung (Teil I)

Definition von Korpus (sächlich, Plural: Korpora) nach H.Bussmann (1990) "Lexikon der Sprachwissenschaft":
"*Endliche Menge von konkreten sprachlichen Äusserungen, die als empirische Grundlage für sprachwissenschaftliche Untersuchungen dienen*."

Ein Textkorpus kann als eine Folge von Wörtern (normale Wörter und Interpunktion) aufgefasst werden. In Prolog kann das durch Fakten der Form

wort(Position, Wort)

repräsentiert werden. Der Beispieltext

Wenn Fliegen hinter Fliegen fliegen, fliegen Fliegen Fliegen nach.

sieht in Prolog dann so aus:

```
wort(1, 'Wenn').
wort(2, 'Fliegen').
wort(3, hinter).
wort(4, 'Fliegen').
wort(5, fliegen).
wort(6, ',').
wort(7, fliegen).
wort(8, 'Fliegen').
wort(9, 'Fliegen').
wort(10, nach).
wort(11, '.').
```

Implementiere folgende Prädikate und teste sie an Hand des Beispieltexts aus.

a) Definiere das Prädikat **suche_wort/2**, das als 1. Argument ein Wort nimmt und als 2. Argument die Position im Text liefert, in der das Wort vorkommt. Wenn ein Wort mehrmals vorkommt, sollen alle Positionen über Backtracking ausgegeben werden.

b) Bigramme nennt man alle direkt nebeneinander liegenden Wörterpaare eines Textes. Definiere das Prädikat **bigramm/2**, das nacheinander durch Backtracking alle Bigramme berechnet. 1. Argument ist 1. Wort des Bigramms, 2. Argument ist 2. Wort des Bigramms.

c) Definiere das Prädikat **wort_liegt_zwischen/3**, das genau dann wahr ist, wenn ein Wort zwischen zwei Positionen liegt, und folgende Argumente nimmt:

1. Argument: Wort
2. Argument: die erste mögliche Position
3. Argument: die letzte möglich Position

d) Vorkommen im Kontext: Oft interessiert, ob ein Wort innerhalb eines bestimmten Kontexts eines andern Worts vorkommt. Definiere das Prädikat **erscheint_im_kontext/3**, das folgende Argument hat:

1. Argument: Wort, in dessen Kontext gesucht werden soll
2. Argument: Anzahl Wörter, die links und rechts vom gesuchten Wort berücksichtigt werden sollen
3. Argument: Wort, das im Kontext vorkommt

Dein Programm sollte nicht nur ja oder nein sagen, sondern alle Wörter innerhalb eines bestimmten Kontext über Backtracking aufzählen können.