Lehrstuhl für Softwaretechnik und Programmiersprachen
Professor Dr. Michael Leuschel

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Einführung in die logische Programmierung - WS 2012/2013
Übungsblatt 4

Bei Fragen bezüglich der Übungen melden Sie sich bitte bei Sven Hager
(sven.hager@hhu.de)

Information Auf der Webseite der Veranstaltung können Sie sich Testfälle für die untenstehenden Aufgaben herunterladen.

Aufgabe 1 - Bäume und Heaps

Diese Aufgabe ist ein weiteres Übungspaket zur Traversierung von Prolog Datenstrukturen.

a) Implementieren Sie das Prädikat `count_leaves(BinTree, NumLeafs)` auf Basis der geordneten Binärbäume von Blatt 3.

b) Schreiben Sie das Prädikat `extract_subtree(BinTree, Key, SubTree)` auf Basis der geordneten Binärbäume von Blatt 3. `SubTree` soll ein Teilbaum von `BinTree` mit Wurzel `Key` sein.

c) Testen Sie mithilfe eines Prädikats `is_heap(Candidate)`, ob `Candidate` ein binärer Maxheap ist. Die Darstellung von `Candidate` soll ebenfalls der der Binärbäume entsprechen, allerdings ohne die Eigenschaft der Sortiertheit.

Aufgabe 2 - Ivo's Challenge

Diese Aufgabe soll ein tieferes Verständnis für die Struktur und Manipulation von Prolog Termen schaffen.

a) Machen Sie sich mit den Prolog Builtinprädikaten `=../2`, `functor/3`, `arg/3`, `var/1`, `nonvar/1`, `term_variables/2` und `==/2` vertraut. Falls Sie SWI Prolog nutzen, können Sie hierfür z.B. den `help` Befehl verwenden. Machen Sie sich insbesondere den Unterschied zwischen `=/2` und `==/2` klar.

b) Implementieren Sie das Prädikat `my_univ(A, B)`, welches sich wie `A =.. B` verhalten soll. Nutzen Sie hierfür die Prädikate `functor/3` und `arg/3`. Falls `A =.. B` einen Fehler wirft, so soll `my_univ(A, B)` einfach fehlschlagen.

c) Implementieren Sie das Prädikat `my_eq(A, B)`, welches sich wie `A == B` verhalten soll. Achten Sie darauf, dass ungebundene Variablen, welche sich in `A` oder `B` befinden, nach dem Test weiterhin ungebunden sind (Tip: Verwenden Sie `term_variables/2`).

Aufgabe 3 - Funktionale Programmierung

In dieser Aufgabe soll es darum gehen, einige Higher Order Funktionen aus der funktionalen Programmierung in Prolog abzubilden.

a) Machen Sie sich mit dem Builtinprädikat `call(Goal)` vertraut. `Goal` ist hier ein beliebiger Prolog Term, welcher als ein Prädikat aufgerufen wird.

b) Schreiben die das Higher Order Prädikat `compose_and_evaluate(F, G, Arg, Result)`, welches die Namen zweier zweistelliger Prädikate `F` und `G` sowie ein Argument `Arg` nimmt und `Result` gemäß

$$(F \circ G)(Arg) = Result$$

berechnet. Hierbei ist $F \circ G$ die Komposition der Funktionen F und G.

Beispiel im mathematischer Notation:

$$f(x) := x^2$$
$$g(x) := x + 1$$
$$\Rightarrow (f \circ g)(4) = f(g(4)) = f(5) = 25$$

c) Implementieren Sie die Higher Order Funktion `map(Function, ArgList, ResultList)`, welches den Namen eines zweistelligen Prädikats `Function` und eine Liste von Argumenten `Arglist` nimmt. `ResultList` soll eine Liste mit Funktionswerten von `Function` ausgewertet an den gegebenen Argumenten in `ArgList` sein.

Beispiel in funktionaler Syntax:

$$map(\lambda\ x \rightarrow x^2, [1, 2, 3]) = [1, 4, 9]$$

d) Implementieren Sie die Higher Order Funktion `reduce(Function, Init, Args, Result)`. `reduce/4` nimmt den Namen einer dreistelligen Funktion `Function`, einen initialen Status `Init` sowie eine Liste `Args` von Argumenten und berechnet schrittweise einen neuen Wert aus dem aktuellen Status sowie dem aktuellen Listenkopf.

Beispiel in functionaler Syntax:

$$\begin{aligned} & reduce(\lambda\ x\ y \rightarrow x + y, 0, [1, 2, 3]) \\ = & reduce(\lambda\ x\ y \rightarrow x + y, 1, [2, 3]) \\ = & reduce(\lambda\ x\ y \rightarrow x + y, 3, [3]) \\ = & reduce(\lambda\ x\ y \rightarrow x + y, 6, []) \\ = & 6 \end{aligned}$$

Aufgabe 4 - Resolution

Betrachten Sie die folgende Prolog Datenbank

```
a.
b.
c.
d :- a, b, e.
e :- h, g.
h.
g.
x :- y, z.
y.
z.
```

Zeigen Sie mithilfe der Resolutionsmethode, dass

$$d \wedge x$$

gilt.