Martin-Luther-Universität Halle-Wittenberg
Naturwissenschaftliche Fakultät III
Institut für Informatik

2. Übung zur Vorlesung „Deduktive Datenbanken und logische Programmierung“

Wintersemester 2007/2008 — Ausgabe: 2007-10-22

Abgabe: 2007-10-29

Aufgabe 2.1: Mathematiker-Stammbaum

a) Gibt es Doktoranden, die vor ihrem Doktorvater promoviert wurden?

b) Berechnen Sie zu jedem mittelbaren Doktorvater von Harold Stephen Finkelstein die Anzahl der dazwischenliegenden akademischen Generationen. Beispiel: Zwischen Finkelstein und Finkelstein liegen null Generationen, zwischen Finkelstein und seinen direkten Doktorvätern liegt eine Generation.
Hinweis: Die Aussage

```
Z is X+Y
```

bedeutet, dass die Summe rechts von `is` berechnet und das Ergebnis mit der linken Seite abgeglichen (sog. „unification“) wird.

c) Schreiben Sie ein Prädikat für Doktoranden mit mehr als einem direkten Doktorvater.

d) Schreiben Sie ein Prädikat für die Anzahl der (direkten) Doktoranden eines Doktorvaters. Benutzen Sie hierzu die Aussageformen

```
length(List, Len)
findall(X, pred(X), ListOfMatchingXs)
```

e) Finden Sie den Doktorvater mit den meisten (direkten) Doktoranden.

Tipp: Nutzen Sie ein Hilfsprädikat für die Doktorväter, auf die das nicht zutrifft.

Falls Ihr Rechner zu lange für diese Aufgabe braucht, überlegen Sie sich, wie Sie die Aufgabe interaktiv im Prolog-Interpreter lösen können.

f) Denken Sie sich selbst eine Anfrage aus, die Sie in Prolog programmieren.

Aufgabe 2.2: Typabgleich

Das Typsystem der Programmiersprache Haskell kennt Typvariablen und kann selbstständig deren Belegung ermitteln. Beispielsweise wendet die Funktion `map` eine andere Funktion elementweise auf Listenelemente an und die Funktion `toUpper` wandelt Buchstaben in Großbuchstaben um. Folglich wandelt `map toUpper` eine Liste von Buchstaben in eine Liste von Großbuchstaben um. Aus den Typsignaturen

```
map :: (a -> b) -> ([a] -> [b])
toUpper :: Char -> Char
```

schließt das Typsystem selbständig, dass `map toUpper` den Typ `[Char] -> [Char]` besitzt. Die Typvariablen `a` und `b` werden beide mit dem konkreten Typ `Char` belegt.
Diese Typbestimmung lässt sich direkt in Prolog übersetzen. Die Typsignaturen werden wie folgt umschrieben

```
typeMap(func(func(X,Y), func(list(X),list(Y)))).
typeToUpper(func(char, char)).
```

und die Abfrage, welche konkreten Typen sich ergeben, könnte zum Beispiel so aussehen:

```
?- typeMap(func(F, G)), typeToUpper(F).

F = func(char, char)
G = func(list(char), list(char)) ;
```

a) Definieren Sie Prädikate für die Typen folgender Funktionen:

- `reverse` – kehrt die Reihenfolge der Elemente einer Liste um
- `append` – hängt zwei Listen aneinander (benutzen Sie die Schönfinkel/Curry-Form)
- `concat` – hat als Parameter eine Liste von Listen und hängt diese aneinander
- `sum` – addiert eine Liste ganzer Zahlen

Führen Sie falls nötig neue Atome für Typen oder Typkonstruktoren ein.

b) Bestimmen Sie mit Prolog die Typen folgender Ausdrücke:

- `map sum`
- `map reverse`
- `sum (map toUpper x)`

c) Bestimmen Sie mit Prolog die Belegung der Typvariablen der verwendeten Funktionen in folgenden Ausdrücken:

- `map toUpper (concat x)`
- `append x (reverse x)`
- `sum (reverse x)`