INFORMATIK

Universität
des
Saarlandes
FR Informatik

Harald Ganzinger
Andreas Podelski
Uwe Waldmann

25. April 2001

Übungen zur Vorlesung „Programmiersprachen“
Übungsblatt 3

Programmieraufgabe 3.1: (*2 P.*)
Wir betrachten die Menge aller Terme, die aus Variablen x, Applikation $(e_1\ e_2)$, und λ-Ausdrücken (`fn` x `=>` e) aufgebaut sind. Diese Terme können wir mittels des folgenden ML-Datentyps repräsentieren:

```
datatype term = var of string
              | app of term * term
              | lambda of string * term;
```

Der Term `x y (fn x => x z)` (der in der Logik gewöhnlich als $(x\,y)(\lambda x.(x\,z))$ geschrieben wird) würde beispielsweise durch

```
app(app(var "x",var "y"),lambda("x",app(var "x",var "z")))
```

codiert.

Schreiben Sie eine ML-Funktion `freeVars : term -> string list`, die die Liste aller Variablen(-namen) liefert, die frei in einem Term auftreten (ohne Duplikate). Beachten Sie, daß eine Variable in einem Term sowohl frei als auch gebunden auftreten kann (wie z.B. x in dem oben angegebenen Term).

Programmieraufgabe 3.2: (*8 P.*)
Implemetieren Sie folgende Funktionen in ML:

- `renameBoundVars : term -> term` benennt die gebundenen Variablen in einem Term so um, daß keine gebundene Variable den gleichen Namen wie eine freie oder eine an anderer Stelle gebundene Variable hat. Aus dem Term

  ```
  app(app(var "x",lambda("x",var "x")),lambda("x",var "x"))
  ```

  könnte also beispielsweise

  ```
  app(app(var "x",lambda("x'",var "x'")),lambda("x''",var "x''"))
  ```

  werden.

– Die Funktion `applyBeta : term -> term -> term` bekommt zwei Argumente t und t'. Falls t ein λ-Ausdruck der Form `lambda(`x`,`e`)` ist, dann liefert `applyBeta` das Ergebnis der β-Reduktion der Applikation $(t\ t')$, das heißt, den Term $e[t'/x]$, der aus e durch Ersetzung von t' für alle freien Auftreten von x in e entsteht. Gebundene Variable in den beteiligten Termen dürfen (und müssen möglicherweise) umbenannt werden. Falls t kein λ-Ausdruck ist, liefert `applyBeta` den Term `app(`t`,`t'`)`.

Aufgabe 3.3: (*3 P.*)
Gegeben sei das folgende ML-Programm:

```
fun insert x [] = [x]
  | insert x (y::l) = if x < y then x::y::l else y::(insert x l);

fun inssort [] = []
  | inssort (x::l) = insert x (inssort l);
```

Zeigen Sie durch strukturelle Induktion, daß die Auswertung von `inssort` l für jede beliebige `int`-Liste l terminiert.

Aufgabe 3.4: (*4 P.*)
Zeigen Sie durch strukturelle Induktion, daß die Auswertung von `inssort` l für jede beliebige `int`-Liste l eine Permutation l' von l liefert, d. h., daß die Multimenge aller Elemente der Liste l mit der Multimenge aller Elemente der Liste l' übereinstimmt. (Multimengen unterscheiden sich von Mengen dadurch, daß Elemente mehrfach vorkommen dürfen.)

Aufgabe 3.5: (*5 P.*)
Zeigen Sie durch strukturelle Induktion, daß die Auswertung von `inssort` l für jede beliebige `int`-Liste l eine sortierte Liste l' liefert, d. h., daß $x \leq y$ für jedes Paar aufeinanderfolgender Elemente x und y in l' gilt.

Schicken Sie die ausgetesteten Lösungen der Programmieraufgaben (inklusive einiger Testbeispiele mit Ergebnissen) bis zum 2. 5., 13:00 Uhr per Mail an Ihren Übungsgruppenleiter, Subject: `PS Blatt 3`. Bereiten Sie die übrigen Aufgaben so vor, daß Sie die Lösung in der nächsten darauffolgenden Übungsstunde vortragen können.