5. Übungsblatt zu Programmierung 1, WS 2012/13

Prof. Dr. Gert Smolka, Sigurd Schneider, B.Sc.
www.ps.uni-sb.de/courses/prog-ws12/

Lesen Sie im Buch: Kapitel 5

Aufgabe 3.30 Betrachten Sie den Ausdruck

```
(fn x => (fn y => (fn x => y) x) y) x
```

a) Geben Sie die Baumdarstellung des Ausdrucks an.
b) Markieren Sie die definierenden Bezeichnerauftreten durch Überstreichen.
c) Stellen Sie die lexikalischen Bindungen durch Pfeile dar.
d) Geben Sie alle Bezeichner an, die in dem Ausdruck frei auftreten.
e) Bereinigen Sie den Ausdruck durch Indizieren der gebundenen Bezeichnerauftreten.

Aufgabe 3.32 Betrachten Sie die bereinigte Deklaration

```
fun f (x, y) = (fn z => (fn u => (fn v => u) z) y) x
```

a) Geben Sie die Baumdarstellung der Deklaration an.
b) Stellen Sie die lexikalischen Bindungen der Deklaration durch Pfeile dar.
c) Geben Sie die statischen Bezeichnerbindungen an, die durch die semantische Analyse bestimmt werden.
d) Geben Sie eine möglichst einfache, semantisch äquivalente Deklaration an, die ohne Abstraktionen gebildet ist.

Zusatzaufgabe Z4.1 Schreiben Sie eine polymorphe Prozedur *member* : $''a \to ''a\ list \to bool$ die testet, ob ein Wert als Element in einer Liste vorkommt. Lösen Sie dies auf drei Arten:

a) durch eine regelbasierte Prozedurdeklaration (formulieren Sie zunächst passende Rekursionsgleichungen),
b) mithilfe der vordeklarierten Prozedur *List.exists*,
c) mithilfe der Prozedur *foldl*.

Aufgabe 4.10 (Count) Schreiben Sie mithilfe von *foldl* eine polymorphe Prozedur *count* : $''a \to ''a\ list \to int$, die zählt, wie oft ein Wert in einer Liste als Element vorkommt. Beispielsweise soll $count\ 5\ [2,5,3,5] = 2$ gelten.

Zusatzaufgabe Z4.2 Schreiben Sie mithilfe der Prozedur *iter* eine Prozedur des Typs $tab : int \to (int \to \alpha) \to \alpha\ list$, die für Argumente n und f das Ergebnis $[f\ 0, \ldots, f\ (n-1)]$ liefert.

Aufgabe 5.2 Schreiben Sie eine Prozedur $sorted : int\ list \to bool$, die testet, ob eine Liste aufsteigend sortiert ist. Verwenden Sie dabei keine Hilfsprozedur.

Aufgabe 5.3 Schreiben Sie eine Prozedur $perm : int\ list \to int\ list \to bool$, die testet, ob zwei Listen bis auf die Anordnung ihrer Elemente gleich sind. Verwenden Sie dabei die Prozedur *isort* und die Tatsache, dass *int list* ein Typ mit Gleichheit ist.

Aufgabe 5.4 Schreiben Sie eine Prozedur *issort* : *int list* → *int list*, die eine Liste sortiert und dabei Mehrfachauftreten von Elementen eliminiert. Beispielsweise soll für $[3,1,3,1,0]$ die Liste $[0,1,3]$ geliefert werden.

Aufgabe 5.9 Deklarieren Sie eine Prozedur

$$\mathit{lex}\colon\ (\alpha * \alpha \to \mathit{order})\ \to\ (\beta * \beta \to \mathit{order})\ \to\ (\alpha * \beta) * (\alpha * \beta) \to \mathit{order}$$

die zu Ordnungen für die Typen α und β die lexikalische Ordnung für den Produkttyp $\alpha * \beta$ liefert. Deklarieren Sie mit Ihrer Prozedur *lex* eine Prozedur, die die lexikalische Ordnung für Paare des Typs *int* $*$ *real* darstellt, die in der ersten Position absteigend und in der zweiten Position aufsteigend sortiert. Zum Beispiel soll $[(3,4.0),\ (2,2.0),\ (2,3.0)]$ gemäß dieser Ordnung sortiert sein.

Aufgabe 5.15 (Striktes Sortieren durch Mischen)

a) Schreiben Sie eine polymorphe Prozedur *smerge*, die zwei strikt sortierte Listen zu einer strikt sortierten Liste kombiniert. Beispielsweise soll gelten: *smerge Int.compare* $([1,3],\ [2,3]) = [1,2,3]$.
b) Schreiben Sie eine polymorphe Prozedur *ssort*, die Listen strikt sortiert. Beispielsweise soll *ssort Int.compare* $[5,3,2,5] = [2,3,5]$ gelten.
c) Machen Sie sich klar, dass es sich bei *ssort Int.compare* um eine Prozedur handelt, die zu einer Liste *xs* die eindeutig bestimmte strikt sortierte Liste liefert, die dieselbe Menge wie *xs* darstellt.

Aufgabe 5.16 Seien endliche Mengen ganzer Zahlen gemäß *Int.compare* durch strikt sortierte Listen dargestellt. Deklarieren Sie Prozeduren wie folgt:

a) *member* : *int* → *int list* → *bool* testet, ob eine Zahl Element einer Menge ist.
b) *union* : *int list* → *int list* → *int list* liefert die Vereinigung zweier Mengen.
c) *intersection* : *int list* → *int list* → *int list* liefert den Schnitt zweier Mengen.
d) *difference* : *int list* → *int list* → *int list* liefert die Differenz zweier Mengen.
e) *eqset* : *int list* → *int list* → *bool* testet, ob zwei Mengen gleich sind.
f) *subset* : *int list* → *int list* → *bool* testet für zwei Mengen X und Y, ob $X \subseteq Y$.
   Realisieren Sie *subset*
   (i) mit *member*
   (ii) mit *eqset* und *intersection* gemäß der Äquivalenz $X \subseteq Y \iff X = X \cap Y$.
   (iii) ohne Hilfsprozedur

Aufgabe 5.18 (Potenzmenge) Seien endliche Mengen durch Listen ohne Mehrfachauftreten dargestellt. Deklarieren Sie eine Prozedur *power* : α *list* → α *list list*, die zu einer Menge X die Menge liefert, die genau die Teilmengen von X enthält. Dabei ist die Reihenfolge beliebig, aber es soll keine Mehrfachauftreten geben. Beispielsweise wäre es korrekt, wenn *power* für $[1,2]$ die Liste $[[1,2],[2],[1],[]]$ liefert.

Aufgabe 5.20 (Liste der ersten n Primzahlen) Deklarieren Sie eine Prozedur *primelist* : *int* → *int list*, die zu $n \geq 0$ die Liste der ersten n Primzahlen in aufsteigender Ordnung liefert. Verwenden Sie die Prozedur *nextprime* aus § 3.11. Hinweis: Verwenden Sie ein Akku und berechnen Sie die Liste der ersten n Primzahlen ab einer gegebenen Primzahl.