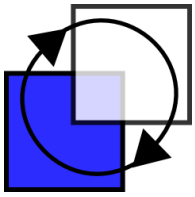
REACTIVE SYSTEMS GROUP

Universität des Saarlandes

Prof. Bernd Finkbeiner, Ph.D.
Markus Rabe, M.Sc.



Programmierung 1 (SS 2010) - 13. Übungsblatt

http://react.cs.uni-saarland.de/prog1/

Lesen Sie Kapitel 13.3 bis 14.5

Aufgabe 13.1

Sei die folgende phrasale Syntax für die Typen von F gegeben:

$$ty \ ::= \ \text{"}\texttt{bool}\text{"} \ | \ \text{"}\texttt{int}\text{"} \ | \ \text{"}\texttt{->}\text{"} \ ty \ ty$$

Die Baumdarstellungen der Typen sollen in Standard ML durch Werte des Typs

```
datatype ty  = Bool | Int | Arrow of ty * ty
```

dargestellt werden, die erforderlichen Wörter durch Werte des Typs *token* aus Kapitel 13.1.

a) Deklarieren Sie einen Prüfer $test: token\ list \to token\ list$.

b) Deklarieren Sie einen Parser $parse: token\ list \to ty * token\ list$.

c) Deklarieren Sie eine Prozedur $rep: ty \to token\ list$, die Typen als Wortfolgen darstellt.

d) Deklarieren Sie eine Prozedur $str: ty \to string$, die Typen als Zeichenfolgen darstellt.

Aufgabe 13.2

Schreiben Sie die Prozedur *lexId* so um, dass sie die folgenden Bezeichner erkennt:

$$\begin{aligned} id &::= letter\ [\, id' \,] \\ id' &::= (\, digit \ | \ letter \,)\ [\, id' \,] \end{aligned}$$

Bezeichner gemäß dieser Grammatik müssen mit einem Buchstaben begonnen werden und können mit Buchstaben und Ziffern fortgesetzt werden.

Aufgabe 13.3

Wir betrachten Ausdrücke, die mit Bezeichnern und den Operatoren :: und @ gebildet werden. Die phrasale Syntax sei durch die Grammatik

$$\begin{aligned} \mathit{exp} &::= \mathit{pexp}\ [\,(\text{“::”} \mid \text{“@”})\ \mathit{exp}\,] \\ \mathit{pexp} &::= \mathit{id} \mid \text{“(”}\ \mathit{exp}\ \text{“)”} \end{aligned}$$

gegeben. Die Operatoren :: und @ klammern also so wie in Standard ML gleichberechtigt rechts: $x :: y \mathbin{@} z :: u \mathbin{@} v \ \rightsquigarrow\ x :: (y \mathbin{@} (z :: (u \mathbin{@} v)))$. Wörter und Baumdarstellungen seien wie folgt dargestellt:

```
datatype token = ID of string | CONS | APPEND | LPAR | RPAR
datatype exp = Id of string | Cons of exp * exp | Append of exp * exp
```

a) Schreiben Sie einen Lexer $\mathit{lex} : \mathit{char\ list} \to \mathit{token\ list}$.

b) Schreiben Sie einen Parser für *exp*.

c) Schreiben Sie eine Prozedur $\mathit{exp} : \mathit{exp} \to \mathit{string}$, die Ausdrücke gemäß der obigen Grammatik mit minimaler Klammerung darstellt.

Aufgabe 13.4

Betrachten Sie applikative Ausdrücke mit der folgenden abstrakten Syntax:

```
datatype exp = Id  of string          (* Identifier  *)
             | App of exp * exp       (* Application *)
```

a) Geben Sie eine eindeutige Grammatik an, die eine phrasale Syntax für diese Ausdrücke beschreibt. Wie in Standard ML soll redundante Klammerung erlaubt sein und Applikationen sollen bei fehlender Klammerung links geklammert werden. Verwenden Sie die Kategorien *exp*, *pexp* und *id*.

b) Schreiben Sie eine Prozedur $\mathit{exp} : \mathit{exp} \to \mathit{string}$, die applikative Ausdrücke gemäß Ihrer Grammatik mit minimaler Klammerung darstellt.

c) Geben Sie eine rechtsrekursive Grammatik mit einer Hilfskategorie exp' an, aus der sich die Parsingprozeduren ableiten lassen.

d) Schreiben Sie eine Prozedur $\mathit{test} : \mathit{token\ list} \to \mathit{bool}$, die testet, ob eine Liste von Wörtern einen applikativen Ausdruck gemäß Ihrer Grammatik darstellt. Wörter sollen wie folgt dargestellt werden:

```
datatype token = ID of string | LPAR | RPAR
```

Aufgabe 13.5

Sei die Struktur *ISet* wie in Abbildung 14.1 im Buch gegeben.

a) Schreiben Sie eine Prozedur $member : int \to ISet.set \to bool$, die testet, ob eine Zahl in einer Menge enthalten ist.

b) Schreiben Sie eine Prozedur $empty : ISet.set \to bool$, die testet, ob eine Menge leer ist.

c) Schreiben Sie eine Prozedur $equal : ISet.set \to ISet.set \to bool$, die testet, ob zwei Mengen gleich sind.

Aufgabe 13.6

Warum ist die folgende Deklaration unzulässig?

```
structure S :> sig eqtype t end = struct
    type t = int -> int
end
```

Aufgabe 13.7

Unter einer Umgebung wollen wir eine Funktion verstehen, deren Definitionsbereich eine Menge von Strings ist. Deklarieren Sie eine Struktur *Env*, die Umgebungen wie folgt implementiert:

$$
\begin{array}{l}
type\ \alpha\ env \\
exception\ Unbound \\
val\ env : (string * \alpha)\ list \to \alpha\ env \\
val\ lookup : \alpha\ env \to string \to \alpha \\
val\ update : \alpha\ env \to string \to \alpha \to \alpha\ env
\end{array}
$$

- *env* liefert zu einer Liste von Paaren die entsprechende Umgebung. Wenn die Liste zu einem String mehr als ein Paar enthält, soll nur eines der Paare verwendet werden.
- *lookup* liefert zu einer Umgebung f und einem String s den Wert fs. Falls f auf s nicht definiert ist, wirft *lookup* die Ausnahme *Unbound*.
- *update* liefert zu einer Umgebung f, einem String s und einem Wert x die Umgebung $f[s := x]$.

Aufgabe 13.8

a) Schreiben Sie eine Prozedur $vector2list : \alpha\ vector \to \alpha\ list$, die zu einem Vektor die entsprechende Liste liefert.

b) Schreiben Sie eine Prozedur $cons : \alpha \to \alpha\ vector \to \alpha\ vector$, die zu x und $[x_1, ..., x_n]$ den Vektor $[x, x_1, ..., x_n]$ liefert. Verwenden Sie dabei *Vector.concat*.

Aufgabe 13.9

Mit Vektoren lassen sich Funktionen, deren Definitionsbereich ein endliches Intervall der ganzen Zahlen ist, mit konstanter Laufzeit berechnen. Schreiben Sie eine Prozedur

$$\mathit{vectorize} : (\mathit{int} \to \alpha) \to \mathit{int} \to \mathit{int} \to \mathit{int} \to \alpha,$$

die zu einer Prozedur p und zwei Zahlen m, n eine Prozedur mit konstanter Laufzeit liefert, die für alle Zahlen $\{x \in \mathbb{Z} \mid m \leq x \leq n\}$ dasselbe Ergebnis wie p liefert und ansonsten die Ausnahme *Subscript* wirft. Nehmen Sie dabei an, dass p auf allen Zahlen zwischen m und n terminiert. *Hinweis:* Schreiben Sie die Prozedur *vectorize* so, dass sie zunächst den Vektor $[p\,m\,, \ldots\,, p\,n]$ bestimmt.

Aufgabe 13.10

Geben Sie Argumente für die Prozedur *position* aus Kapitel 14.5 an, für die die Prozedur maximale Laufzeit hat. Verwenden Sie dabei Listen der Längen 4 und 7, und geben Sie für beide Fälle ein verkürztes Ausführungsprotokoll an.

Aufgabe 13.11

Deklarieren Sie eine Struktur *IVSet*, die Mengen ganzer Zahlen mit strikt sortierten Vektoren gemäß der folgenden Signatur implementiert:

$$\begin{array}{l} \mathit{eqtype\ set} \\ \mathit{val\ set} : \mathit{int\ list} \to \mathit{set} \\ \mathit{val\ subset} : \mathit{set} \to \mathit{set} \to \mathit{bool} \\ \mathit{val\ elem} : \mathit{int} \to \mathit{set} \to \mathit{bool} \end{array}$$

Dabei soll *set* linear-logarithmische, *subset* lineare und der Elementtest *elem* logarithmische Komplexität haben.