12. Übungsblatt zu Programmierung

Prof. Gert Smolka, Thorsten Brunklaus
www.ps.uni-sb.de/courses/prog-ws00/

Abgabe: 9. Februar 2001 vor der Vorlesung (teils per Email und teils auf Papier)

Allgemeine Hinweise: Die Übungsblätter sollen in Zweiergruppen bearbeitet werden. Die Lösungen der Aufgaben 12.1 und 12.2 geben Sie bitte auf Papier ab. Alle anderen Lösungen schicken Sie bitte bis Freitag vor der Vorlesung per Email an Ihren Übungsgruppenleiter. Jede Gruppe soll nur eine Lösung einreichen, versehen mit den Namen und den Matrikelnummern der Gruppenmitglieder.

Aufgabe 12.1: Syntaxbäume (2+2+2)

(a) Zeichnen Sie einen Syntaxbaum für den Satz

```
(bool -> int -> bool) -> int
```

gemäß der Grammatik für Typen in Abschnitt 12.2. Wie viele solcher Syntaxbäume gibt es?

(b) Zeichnen Sie einen Syntaxbaum für den Satz

```
3*(2*y+x)*z
```

gemäß der linksrekursiven Grammatik für arithmetische Ausdrücke in Abbildung 12.7. Wie viele solcher Syntaxbäume gibt es?

(c) Zeichnen Sie je einen Syntaxbaum für den Satz `2+3-4+7` gemäß

(i) der linksrekursiven Grammatik für arithmetische Ausdrücke in Abbildung 12.7.
(ii) der rechtsrekursiven Grammatik für arithmetische Ausdrücke in Abbildung 12.9.
(iii) der modifizierten rechtsrekursiven Grammatik für arithmetische Ausdrücke in Abbildung 12.10.

Aufgabe 12.2: Mehrdeutige Grammatik (8) Zeigen Sie mit einem Gegenbeispiel, dass die kontextfreie Grammatik

exp = "x" | *exp exp*

nicht eindeutig ist. Geben Sie eine eindeutige Grammatik an, die genau dieselben Sätze darstellt.

Aufgabe 12.3: Darstellung von Typen (8) Gegeben sei die Typdeklaration

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

Schreiben Sie eine Prozedur

```
ty : ty -> string
```

die Typen durch Zeichenreihen darstellt. Die Darstellung soll gemäß der kontextfreien Grammatik in Abschnitt 12.2 erfolgen und möglichst wenig Klammern und Trennzeichen enthalten. Führen Sie für jedes Nonterminal der Grammatik eine Hilfsprozedur ein.

Aufgabe 12.4: Darstellung von Ausdrücken (8) Gegeben sei die Typdeklarationen

```
datatype ops = Add | Sub | Mul | Leq

datatype exp = Con of int | Op of exp * ops * exp
```

Schreiben Sie eine Prozedur

```
exp : exp -> string
```

die Ausdrücke durch Zeichenreihen darstellt. Die Darstellung soll gemäß der linksrekursiven Grammatik in Abbildung 12.7 erfolgen und möglichst wenig Klammern und Trennzeichen enthalten. Führen Sie für jedes Nonterminal der Grammatik eine Hilfsprozedur ein.

Aufgabe 12.5: Ausdrücke mit Cons und Applikation (8+4+8) Wir betrachten Ausdrücke, die mit Bezeichnern, Applikation (Prozeduranwendung) und den Listenoperationen :: (Cons) und @ (Append) gebildet werden können. Wir verwenden die abstrakte Syntax

```
datatype ops = Cons | Append

datatype exp = Id of string
             | Op of exp * ops * exp
             | App of exp * exp
```

Die konkrete Syntax der Ausdrücke soll der von Standard ML entsprechen. Dabei stehen die Infixoperatoren `::` und `@` auf derselben Rangstufe und werden rechtsassoziativ gruppiert. Applikation steht auf der tiefsten Rangstufe und wird linksassoziativ gruppiert. Beispielsweise soll

```
x :: y x x :: z @ u
```

denselben Ausdruck darstellen wie:

```
x :: (((y x) x) :: (z @ u))
```

Die kontextfreie Syntax definieren wir mit der folgenden Grammatik:

$$
\begin{aligned}
exp &= apexp\ [\ (\texttt{"::"}\ \mid\ \texttt{"@"})\ exp\ ] \\
apexp &= [\ apexp\ ]\ atexp \\
atexp &= identifier\ \mid\ \texttt{"("}\ exp\ \texttt{")"}
\end{aligned}
$$

Die lexikalische Syntax besteht aus Bezeichnern und den Schlüsselwörtern `"::"`, `"@"`, `"("` und `")"`. Ein Bezeichner ist eine nichtleere Zeichenreihe, die aus Buchstaben und Ziffern besteht und mit einem Buchstaben anfängt.
Wörter stellen wir mithilfe des folgenden Typs dar:

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

(a) Schreiben Sie eine Prozedur

```
lex : string -> token list  (* Error *)
```

die die durch eine Zeichenreihe dargestellte Wortliste liefert.

(b) Geben Sie eine Grammatik an, die dieselben Sätze wie die obige Grammatik liefert, aber eine Form hat, aus der sich die Parsingprozeduren ableiten lassen.

(c) Schreiben Sie eine Prozedur

```
parse : token list -> exp  (* Error *)
```

die den durch eine Wortliste dargestellten Ausdruck liefert.