9. Übungsblatt zu Programmierung 1, WS 2008/09

Prof. Dr. Gert Smolka, Mark Kaminski, M.Sc.
www.ps.uni-sb.de/courses/prog-ws08/

Lesen Sie im Buch: Kapitel 13

Aufgabe 13.1 Deklarieren Sie eine endrekursive Prozedur

lex' : *token list* → *char list* → *token list*

sodass lex' [] *cs* für jede Zeichenfolge *cs* dasselbe Ergebnis liefert wie *lex cs*.

Aufgabe 13.2 Ändern Sie die Deklaration von *lex* so ab, dass die Wörter *bool* und *int* immer durch mindestens ein Leerzeichen getrennt sein müssen.

Aufgabe 13.3 Zeichnen Sie Syntaxbäume für *ty* und die folgenden Wortfolgen:

a) *int* → (*bool* → *int*)
b) (*bool* → *bool*) → *int*
c) *int* → (*bool* → *bool*) → *int*

Aufgabe 13.4 (Mehrdeutige Grammatik) Zeigen Sie mit einem Beispiel, dass die Grammatik *exp* ::= `"x"` | *exp exp* nicht eindeutig ist. Geben Sie eine eindeutige Grammatik an, die dieselben Wortfolgen darstellt.

Aufgabe 13.5 Die beschriebene Grammatik für die Typen von F liefert für jeden Typ unendlich viele Wortdarstellungen. Sie sollen eindeutige Grammatiken angeben, die für jeden Typ genau eine Wortdarstellung liefern. Dabei soll der Typ (*int* → *int*) → *int* → *int* jeweils wie folgt dargestellt werden:

a) ((*int* → *int*) → (*int* → *int*)) (vollständig geklammert)
b) →→ *int int* → *int int* (Präfixdarstellung ohne Klammern)
c) (*int* → *int*) → *int* → *int* (minimal geklammert) kniffelig!

Aufgabe 13.6 Sei die folgende phrasale Syntax für die Typen von *F* gegeben:

ty ::= `"bool"` | `"int"` | `"->"` *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 § 13.1.

a) Deklarieren Sie einen Prüfer *test* : *token list* → *token list*.
b) Deklarieren Sie einen Parser *parse* : *token list* → *ty* ∗ *token list*.
c) Deklarieren Sie eine Prozedur *rep* : *ty* → *token list*, die Typen als Wortfolgen darstellt.
d) Deklarieren Sie eine Prozedur *str* : *ty* → *string*, die Typen als Zeichenfolgen darstellt.

Aufgabe 13.7 (Baumrekonstruktion aus der Prälinearisierung) Wir können die Prälinearisierung eines Baums (§ 7.6.3) als eine Wortdarstellung des Baums auffassen.

a) Schreiben Sie eine Prozedur *rep* : *tree* → *int list*, die die Prälinearisierung eines Baums liefert.
b) Schreiben Sie eine Prozedur *parse* : *int list* → *tree* ∗ *int list*, die einen Baum aus seiner Prälinearisierung rekonstruiert. Für jeden Baum t soll gelten: $parse(rep\ t) = (t, nil)$.

Aufgabe 13.8 Deklarieren Sie eine Prozedur *ty* : *ty* → *string*, die die Typen von F durch Zeichenfolgen darstellt. Die Darstellung soll gemäß der in § 13.1 und § 13.2 definierten lexikalischen und phrasalen Syntax erfolgen und nur die unbedingt notwendigen Klammern und Leerzeichen enthalten.

Aufgabe 13.11 (Listen) Wir betrachten Ausdrücke, die mit Bezeichnern und den Operatoren :: und @ gebildet werden. Die phrasale Syntax sei durch die Grammatik

$$\begin{aligned} exp &::= pexp\ [\ (\texttt{"::"} \mid \texttt{"@"})\ exp\ ] \\ pexp &::= id \mid \texttt{"("}\ exp\ \texttt{")"} \end{aligned}$$

gegeben. Die Operatoren :: und @ klammern also so wie in Standard ML gleichberechtigt rechts: $x :: y @ z :: u @ v \rightsquigarrow x :: (y @ (z :: (u @ v)))$. 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 *lex* : *char list* → *token list*.
b) Schreiben Sie einen Parser für *exp*.
c) Schreiben Sie eine Prozedur *exp* : *exp* → *string*, die Ausdrücke gemäß der obigen Grammatik mit minimaler Klammerung darstellt.

Aufgabe 13.13 (Typen) Wir betrachten Typen, die mit *bool*, *int*, ∗ und → gebildet werden. Die konkrete Syntax soll der von Standard ML entsprechen. Dabei klammert ∗ vor → und → klammert rechts: $int * int * int \rightarrow int \rightarrow int \rightsquigarrow (int * int * int) \rightarrow (int \rightarrow int)$. Die mit ∗ dargestellten Produkte können zwei- oder höherstellig sein. Wörter und Baumdarstellungen seien wie folgt dargestellt:

```
datatype token = BOOL | INT | STAR | ARROW | LPAR | RPAR

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

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

b) Beschreiben Sie die phrasale Syntax durch eine eindeutige Grammatik, die affin zu der durch den Typ *ty* gegebenen abstrakten Grammatik ist. Verwenden Sie die Kategorien *ty*, *sty* und *pty*.

c) Erweitern Sie die Grammatik um eine Hilfskategorie *sty′*, damit sie sich als Grundlage für Parser eignet. Durch die zusätzliche Kategorie soll erreicht werden, dass die Parser Produkte korrekt übersetzen: $[INT] \rightsquigarrow Int$ und $[INT, STAR, BOOL, STAR, INT] \rightsquigarrow Star[Int, Bool, Int]$.

d) Schreiben Sie Parser für die erweiterte Grammatik.