12. Übungsblatt zu Programmierung WS 2002 / 03

Prof. Dr. Gert Smolka und Dipl.-Inform. Thorsten Brunklaus
`www.ps.uni-sb.de/courses/prog-ws02/`

Abgabe: Montag, 27. Januar 2003

Schicken Sie Ihre Lösungen per E-Mail an Ihren Übungsgruppenleiter. Geben Sie in der E-Mail zuerst ihren Namen und die Nummer Ihrer Übungsgruppe an. Bitte reichen Sie alle Lösungen mit nur einer E-Mail ein.

Die zu zeichnenden Syntaxbäume und Dependenzgraphen sollen nicht eingereicht werden. Die jeweiligen Punkte werden Ihnen ohne Abgabe zuerkannt.

Aufgabe 12.1: Syntaxbäume ($10 = 2 + 2 + 3 * 2$)

(a) Zeichnen Sie einen Syntaxbaum für den Satz

```
(bool → int → bool) → int
```

gemäß der Grammatik für Typen im Skript. 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 im Skript. 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 im Skript.
(ii) der rechtsrekursiven Grammatik für arithmetische Ausdrücke im Skript.
(iii) der modifizierten rechtsrekursiven Grammatik für arithmetische Ausdrücke im Skript.

Aufgabe 12.2: Mehrdeutige Grammatik ($6 = 2 + 4$) Zeigen Sie mit einem Gegenbeispiel, dass die kontextfreie Grammatik

exp = "x" | *exp exp*

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

Aufgabe 12.3: Zeichendarstellung 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 Skript erfolgen und möglichst wenig Klammern und Leerzeichen enthalten. Führen Sie für jedes Nonterminal der Grammatik eine Hilfsprozedur ein.

Aufgabe 12.4: Zeichendarstellung von Ausdrücken (10) Seien die Typdeklarationen

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

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

gegeben. Schreiben Sie eine Prozedur

```
exp : exp → string
```

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

Aufgabe 12.5: Lexer und Parser für Typen mit Pfeil und Stern ($20 = 6 + 6 + 8$) Sie sollen einen Lexer und einen Parser für Typen schreiben, die mit `int`, `->`, `*` und Klammern gebildet werden können. Die konkrete Syntax der Typen soll der von Standard ML entsprechen. Dabei steht `->` auf einer höheren Rangstufe als `*` und wird rechtsassoziativ gruppiert. Beispielsweise soll

```
int * int * int → int * int
```

denselben Typ darstellen wie

```
(int * int * int) → (int * int)
```

Die mit `*` dargestellten Produkte können zwei-, drei- oder höherstellig sein.

(a) Schreiben Sie einen Lexer wie folgt:

```
lex : string → token list   (* Error *)

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

(b) Geben Sie eine eindeutige kontextfreie Grammatik für die Typen an. Halten Sie sich dabei an die obigen Vorgaben. Verwenden Sie die Nonterminale *ty*, *sty* und *aty*. Behandeln sie `*` als rechtsassoziativen Infixoperator. Zeichnen Sie den Dependenzgraphen der Grammatik.

(c) Schreiben Sie einen Parser wie folgt:

```
parse : token list → ty   (* Error *)

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

Die mit `*` dargestellten Produkte sollen so wie in den folgenden Beispielen interpretiert werden:

```
int * int * int       ⟹   Star[Int, Int, Int]
(int * int) * int     ⟹   Star[Star[Int, Int], Int]
int * (int * int)     ⟹   Star[Int, Star[Int, Int]]
```

Die Parsingprozeduren sollen den Nonterminalen Ihrer Grammatik entsprechen. Verwenden Sie die Hilfsprozeduren `match` und `combine`.

Aufgabe 12.6: Kontextfreie Syntax von applikativen Ausdrücken ($16 = 4 * 4$) Sei die folgende abstrakte Syntax für applikative Ausdrücke gegeben:

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

(a) Geben Sie eine eindeutige kontextfreie Grammatik für diese Ausdrücke an, die Applikationen linksassoziativ gruppiert und bei der Teilausdrücke nach Belieben geklammert werden können. Verwenden Sie die Nonterminale *exp*, *atexp* und *identifier*. Nehmen Sie dabei an, dass das Nonterminal *identifier* für Bezeichner bereits definiert ist.

(b) Schreiben Sie eine Prozedur

exp : exp → string

die Ausdrücke gemäß der obigen Grammatik mit möglichst wenigen Klammern darstellt.

(c) Geben Sie eine rechtsrekursive Grammatik mit Hilfsterminal *apexp*′ an, aus der sich die Parsingprozeduren ableiten lassen. Zeichnen Sie den Dependenzgraphen der Grammatik.

(d) Schreiben Sie eine Prozedur

test : token list → bool

die testet, ob eine Liste von Wörtern einen Ausdruck gemäß der obigen Grammatik darstellt. Dabei sollen Wörter wie folgt dargestellt sein:

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

Aufgabe 12.7: Lexer und Parser für Ausdrücke mit Cons und Append $(30 = 10 + 6 + 4 + 10)$
Sie sollen einen Lexer und einen Parser für Ausdrücke schreiben, die mit Bezeichnern, Applikation, Klammern und den Listenoperationen Cons (`::`) und Append (`@`) gebildet werden können. 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 lexikalische Syntax der Ausdrücke soll aus Bezeichnern und den Wörtern "`::`", "`@`", "`(`" und "`)`" bestehen. Ein Bezeichner soll eine nichtleere Zeichenreihe sein, die aus Buchstaben und Ziffern besteht und mit einem Buchstaben anfängt.

(a) Schreiben Sie einen Lexer wie folgt:

```
lex : string → token list  (* Error *)

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

(b) Geben Sie eine eindeutige kontextfreie Grammatik für die Ausdrücke an. Halten Sie sich dabei an die obigen Vorgaben. Verwenden Sie die die Nonterminale *exp*, *apexp*, *atexp* und *identifier*. Nehmen Sie dabei an, dass das Nonterminal *identifier* für Bezeichner bereits definiert ist. Zeichnen Sie den Dependenzgraphen der Grammatik.

(c) Geben Sie eine rechtsrekursive Grammatik mit Hilfsterminal *apexp*′ an, aus der sich die Parsingprozeduren ableiten lassen. Zeichnen Sie den Dependenzgraphen der Grammatik.

(d) Schreiben Sie einen Parser wie folgt:

```
parse : token list → exp  (* Error *)

datatype opr = Cons | Append

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