12. Übungsblatt zu Programmierung 1, WS 2012/13

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

Lesen Sie im Buch: Kapitel 13

Aufgabe 12.17 Geben Sie passend zu den Gleichungen in Abbildung 12.3 auf S. 244 eine Gleichung an, die die freien Bezeichner rekursiver Abstraktionen beschreibt.

Aufgabe 12.18 Warum ist die rekursive Abstraktion $\mathit{rfn}\ f\ (f : \mathit{int}) : \mathit{int} \Rightarrow f$ gemäß der Regel Srabs semantisch zulässig?

Aufgabe 12.19 (Erweiterung der Prozeduren elab und eval)

a) Erweitern Sie die Deklaration der Typen *exp* und *value* um einen Konstruktor für rekursive Abstraktionen.
b) Erweitern Sie die Prozedur *elab* um eine Regel für rekursive Abstraktionen.
c) Erweitern Sie die Prozedur *eval* um eine Regel für rekursive Abstraktionen. Erweitern Sie zudem die Fallunterscheidung in der Regel für Prozeduranwendungen um die Anwendung rekursiver Prozeduren. Erproben Sie Ihre erweiterte Prozedur *eval* mit einer rekursiven Prozedur, die Fakultäten berechnet.
d) Geben Sie einen zulässigen Ausdruck *e* an, sodass die Auswertung von *eval empty e* divergiert.

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

a) $\mathit{int} \to (\mathit{bool} \to \mathit{int})$
b) $(\mathit{bool} \to \mathit{bool}) \to \mathit{int}$
c) $\mathit{int} \to (\mathit{bool} \to \mathit{bool}) \to \mathit{int}$

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 $(\mathit{int} \to \mathit{int}) \to \mathit{int} \to \mathit{int}$ jeweils wie folgt darstellt werden:

a) $((\mathit{int} \to \mathit{int}) \to (\mathit{int} \to \mathit{int}))$ (vollständig geklammert)

b) $(int \to int) \to int \to int$ (minimal geklammert)
c) $\to\to int\ int \to int\ int$ (Präfixdarstellung ohne Klammern)

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, und die erforderlichen Wörter durch Werte des Typs *token* aus Abschnitt 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 13.8 Deklarieren Sie eine Prozedur $ty : ty \to string$, die die Typen von F durch Zeichenfolgen darstellt. Die Darstellung soll gemäß der in den Abschnitten 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

exp = $pexp$ [(`"::"` | `"@"`) exp]
$pexp$ = id | `"("` exp `")"`

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 $lex : char\ list \to token\ list$.
b) Schreiben Sie einen Parser für *exp*.
c) Schreiben Sie eine Prozedur $exp : exp \to string$, die Ausdrücke gemäß der obigen Grammatik mit minimaler Klammerung darstellt.

Aufgabe 13.14 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 $exp : exp \rightarrow string$, 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 $test : token\ list \rightarrow bool$, 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
```