8. 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 12

Aufgabe 12.1 Geben Sie Deklarationen an (in Standard ML), die den Bezeichner *e* an die abstrakte Darstellung des Ausdrucks

$$fn\ f : int \to int \Rightarrow\ fn\ n : int \Rightarrow\ if\ n \leq 0\ then\ 1\ else\ n * f(n-1)$$

binden. Gehen Sie dabei schrittweise vor und beginnen Sie mit der Deklaration des Teilausdrucks $n \leq 0$:

```
val e1 = Opr(Leq, Id"n", Con(IC 0))
```

Aufgabe 12.2 In § 2.4 und § 3.8 haben wir definiert, was wir unter offenen und geschlossenen Ausdrücken und den freien Variablen eines Ausdrucks verstehen wollen.

a) Schreiben Sie eine Prozedur $closed : exp \to bool$, die testet, ob ein Ausdruck geschlossen ist. Verwenden Sie dabei eine Hilfsprozedur $closed' : exp \to id\ list \to bool$, die testet, ob alle freien Bezeichner eines Ausdrucks in einer Liste vorkommen.

b) Schreiben Sie eine Prozedur $free : exp \to id\ list$, die zu einem Ausdruck eine Liste liefert, die die in diesem Ausdruck frei auftretenden Bezeichner enthält. Die Liste darf denselben Bezeichner mehrfach enthalten. Verwenden Sie eine Hilfsprozedur $free' : id\ list \to exp \to id\ list$, die nur die frei auftretenden Bezeichner liefert, die nicht in einer Liste von "gebundenen" Bezeichnern enthalten sind.

Aufgabe 12.5 Geben Sie Typumgebungen an, für die der Ausdruck $if\ true\ then\ x\ else\ y$ zulässig beziehungsweise unzulässig ist.

Aufgabe 12.6 Geben Sie eine Ableitung für die folgende Aussage an: $[x := int] \vdash fn\ f : int \to bool \Rightarrow fn\ y : int \Rightarrow f(2 * x + y) : (int \to bool) \to (int \to bool)$.

Aufgabe 12.7 Deklarieren Sie mithilfe der Prozedur *elab* eine Prozedur $test : exp \to bool$, die testet, ob ein Ausdruck geschlossen und zulässig ist.

Aufgabe 12.8 Geben Sie einen möglichst einfachen Ausdruck *e* an, sodass bei der Anwendung von *elab* auf *empty* und *e* jede der 6 Prozedurregeln zum Einsatz kommt. Erproben Sie *elab* mit einem Interpreter für diesen Ausdruck.

Aufgabe 12.9 Die Prozedur *elab* kann durch Werfen einer Ausnahme *Error s* einen Fehler *s* melden. Geben Sie möglichst einfache Ausdrücke an, für die die Prozedur *elab empty* die Fehler "*T Opr*", "*T If1*", "*T If2*", "*T App1*" und "*T App2*" meldet.

Aufgabe 12.11 Sei *e* eine Darstellung des Ausdrucks $fn\ x : int \Rightarrow y$. Überlegen Sie sich, welche Ergebnisse die folgenden Aufrufe von *elab* und *eval* liefern.

a) *elab empty e*
b) *eval empty e*
c) *eval empty* (*App*(*e*, *Con*(*IC 7*)))

Aufgabe 12.12 Geben Sie einen möglichst einfachen Ausdruck *e* an, sodass bei der Ausführung von *eval empty e* jede der 6 Regeln der Prozedur *eval* zum Einsatz kommt.

Aufgabe 12.13 Die Prozedur *eval* kann durch Werfen einer Ausnahme *Error s* einen Fehler *s* melden. Geben Sie möglichst einfache Ausdrücke an, für die die Prozedur *eval empty* die Fehler "*R Opr*", "*R If*" und "*R App*" meldet. Überprüfen Sie Ihre Antworten mit einem Interpreter.

Aufgabe 12.14 Die Prozedur *eval empty* liefert für viele Ausdrücke Ergebnisse, für die *elab empty* Fehler meldet. Geben Sie für jeden der von *elab* behandelten Fehler einen entsprechenden Ausdruck an.

Aufgabe 12.15 (Selbstanwendung und Rekursion) Der Auswertungssatz besagt, dass die Prozedur *eval empty* für jeden zulässigen Ausdruck terminiert. Interessanterweise gibt es unzulässige Ausdrücke (im Sinne der statischen Semantik), deren Auswertung divergiert. Diese können mit dem Ausdruck $fn\ g : t \Rightarrow g\,g$ gebildet werden, der eine Prozedur beschreibt, die ihre Argumentprozedur *g* auf sich selbst anwendet. Dieser Ausdruck ist für jeden Argumenttyp *t* unzulässig.

a) Geben Sie einen (semantisch unzulässigen) Ausdruck an, für den *eval empty* divergiert.
b) Der Logiker Alonzo Church hat um 1930 entdeckt, dass sich prozedurale Rekursion durch Selbstanwendung von Prozeduren simulieren lässt. Versuchen Sie, einen (semantisch unzulässigen) Ausdruck zu finden, der eine Prozedur beschreibt, die die Fakultäten berechnet. Kniffelig!

Aufgabe 12.16 (Paare) Wir wollen F um Paare erweitern. Die abstrakte Syntax und die Menge der Werte erweitern wir wie folgt:

$$t \in Ty = \cdots \mid t * t$$

$$e \in Exp = \cdots \mid (e, e) \mid fst\ e \mid snd\ e$$

$$v \in Val = \mathbb{Z} \cup Pro \cup (Val \times Val)$$

a) Geben Sie die Inferenzregeln für die statische Semantik von Paaren an.
b) Geben Sie die Inferenzregeln für die dynamische Semantik von Paaren an.
c) Erweitern Sie die Deklarationen der Typen *ty*, *exp* und *value* um Konstruktoren für Paare.
d) Erweitern Sie die Deklaration der Prozedur *elab* um Regeln für Paare.
e) Erweitern Sie die Deklaration der Prozedur *eval* um Regeln für Paare.

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 $rfn\ f\ (f : int) : int \Rightarrow f$ 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 12.20 (Deklarationen) Sie wissen jetzt genug, um F in eigener Regie um Deklarationen, Programme und Let-Ausdrücke zu erweitern:

$$d \in Dek = val\ x = e \mid fun\ f\ (x : t) : t' = e$$

$$p \in Prg = d \ldots d$$

$$e \in Exp = \cdots \mid let\ p\ in\ e$$

a) Geben Sie die Inferenzregeln für die dynamische Semantik der neuen Konstrukte an. Orientieren Sie sich dabei an der informellen Beschreibung in § 2.7 und verwenden Sie Aussagen der Form $V \vdash d \triangleright V'$ und $V \vdash p \triangleright V'$.
b) Geben Sie die Inferenzregeln für die statische Semantik der neuen Konstrukte an. Verwenden Sie dabei Aussagen der Form $T \vdash d : T'$ und $T \vdash p : T'$.
c) Implementieren Sie die abstrakte Syntax mit verschränkt rekursiv deklarierten Konstruktortypen *dek*, *prg* und *exp*. Stellen Sie dabei Programme mithilfe von Listen dar.
d) Erweitern Sie die Prozedur *elab* mit verschränkt rekursiv deklarierten Hilfsprozeduren *elabDek* und *elabPrg*.
e) Erweitern Sie die Prozedur *eval* mit verschränkt rekursiv deklarierten Hilfsprozeduren *evalDek* und *evalPrg*.