Prof. Dr. Manfred Schmidt-Schauß
Künstliche Intelligenz/Softwaretechnologie

Fachbereich Informatik (20)
Johann Wolfgang Goethe-Universität Frankfurt am Main

Einführung in das funktionale Programmieren

Wintersemester 1995/96

Übungsblatt Nr. 12

Wie es eine Geometrie im Raume gibt, gibt es auch eine Psychologie in der Zeit, in der die Berechnungen einer Oberflächenpsychologie nicht mehr stimmen würden, weil man darin die Zeit und eine der Formen, die sie annimmt, nämlich das Vergessen, nicht genügend berücksichtigt hätte – das Vergessen, dessen Macht ich zu spüren begann und das ein so gewaltiges Werkzeug der Anpassung an die Wirklichkeit ist, weil es allmählich in uns die überlebende Vergangenheit zerstört, die zu jener in beständigem Widerspruch steht.

aus Marcel Proust: Auf der Suche nach der verlorenen Zeit VI

Aufgabe 1

In der STG-Sprache gibt es nur noch Anwendungen von Funktionen auf Argumente, die syntaktisch als Variablen oder Konstanten vorliegen. Es können keine direkten verschachtelten Anwendungen vorkommen. Solche sind vorher mit einem `let` bzw. `letrec` Konstrukt an eine Variable zu binden.

Beispiel:

```
map f [] = []
map f (x:xs) = (f x):(map f ys)
```

wird in der STG-Sprache zu:

```
map
 = {} \n {f,xs} ->
     case xs of
        Nil {}          -> Nil {}
        Cons {x,xs}     -> let fy  = {f,y}  \u {} f {y}
                               mfy = {f,ys} \u {} map {f,ys}
                           in Cons {fy,mfy}
```

Zur Übersetzung von CORE-FP in die STG-Sprache sind folgende Schritte zu vollbringen:

- Verkettete Anwendungen auf ein Argument sind in eine Anwendung auf mehrere Argumente aufzulösen:
  $(\ldots((f\ e_1)\ldots)e_n \Rightarrow f\{e_1,\ldots,e_n\}$
- Konstruktoren müssen immer gesättigt auf genügend Argumente angewendet werden:
  $c\ \{e,\ldots,e_n\} \Rightarrow \lambda y_1\ldots y_m.c\ \{e,\ldots,e_n,y_1\ldots y_m\}$
  wenn c die Stelligkeit $n+m$ hat.
- alle nicht-atomaren Funktionsargumente sind durch let an eine Variable zu binden.

- es werden in λ-Ausdrücken die freien Variablen und das *update*-Flag gesetzt.

a) Schreiben Sie eine Funktion, die die Liste aller Primzahlen als Ergebnis hat in Gofer.

b) Überstzen Sie ihr Programm aus a) in ein STG-Programm.

Aufgabe 2

In der STG-Sprache sind eingebaute Werte wie Integer auch als *unboxed* Werte implizit benutzbar. Der eigentliche Datentyp Integer ist somit im Prinzip ein algebarischer Datentyp der Art:

```
data Int = Int Int#
```

Hierbei ist `Int#` der Datentyp der eingebauten primitiven Integerzahlen. Es kann jetzt Funktionen geben, die *unboxed* Integer benutzen, und Funktionen, die *boxed* Integer benutzen:

```
plus = {} \n {x,y} ->
     case x of Int {x#}
          -> case y of Int {y#}
                  -> let xy#  = {x#,y#} \u {} -> +# {x#,y#}
                     in  Int {xy#}
```

Hierbei ist `plus` die Funktion, die auf *boxed* Integern operiert und `+#` die Addition auf *unboxed* Integern.

Somit kann man das Laden von Zahlen aus dem Speicher auf einen Basisstack in der STG-Sprache explizit machen.

Schreiben Sie ihr Programm aus Aufgabe 1 um, indem Sie explizites *Unboxing* verwenden. Versuchen Sie das Programm zu optimieren, indem Sie unnötige *Boxing-Unboxing*-Folgen vermeiden.