Lehrstuhl für Programmiersprachen und Übersetzerkonstruktion
CAU Kiel, Institut für Informatik
Prof. Dr. M. Hanus

24118 Kiel
Christan-Albrechts-Platz 4
Tel.: 0431 / 880-7271

3. Übung „Übersetzerbau“

Bearbeitung bis zum 6. Mai 2008

Bitte senden Sie alle Programmieraufgaben zusätzlich zur Abgabe in ausgedruckter Form auch per Email an Axel Stronzik (`axs@informatik.uni-kiel.de`)!

Aufgabe 7
Ein Crosscompiler wird verwendet, um einen Compiler für eine Sprache Q von einer Rechnerarchitektur M_1 auf eine andere Rechnerarchitektur M_2 zu portieren, auf der es noch keinen Compiler für Q gibt. In der Vorlesung wurde die Darstellung von Compilern als T-Diagramme eingeführt. Wir wollen mit dieser Darstellung den Vorgang des Crosscompilierens beschreiben. Gehen Sie davon aus, dass der Compiler für M_1 in ein Frontend $c_f : Q \longrightarrow Z$ und ein Backend $c_b : Z \longrightarrow M_1$ aufgeteilt ist, welche beide in Q geschrieben sind.
Für die Crosscompilierung muss nur das Backend neu implementiert werden. Geben Sie das T-Diagramm für das neue Backend c_b' an und veranschaulichen Sie den Vorgang des Crosscompilierens mit Hilfe von T-Diagrammen.

Aufgabe 8
Für reguläre Ausdrücke sind die folgenden Abkürzungen üblich:

- $[a_1, a_2, \ldots, a_n] := (a_1 \mid a_2 \mid \cdots \mid a_n)$ für $a_1, a_2, \ldots, a_n \in \Sigma$.
- $\alpha^+ := \alpha\alpha^*$ für $\alpha \in \mathrm{RA}(\Sigma)$.
- $\alpha? := (\alpha \mid \varepsilon)$ für $\alpha \in \mathrm{RA}(\Sigma)$.

Erweitern Sie das Verfahren nach Thompson (regulärer Ausdruck $\rightsquigarrow$ NFA) derart, dass für diese Abkürzungen kleinere NFAs erzeugt werden. Wenden Sie Ihre Erweiterungen auf $\beta = [\mathtt{a}, \mathtt{b}, \mathtt{c}]^+\mathtt{d}? \in \mathrm{RA}(\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\})$ an.

Aufgabe 9
Implementieren Sie von Hand einen Scanner für die Programmiersprache Simple in Haskell. Definieren Sie zunächst einen algebraischen Datentyp `Token`, der alle möglichen Symbolklassen enthält. Definieren Sie dann eine Funktion

```
scan :: String -> [Token]
```

Aufgabe 10
Geben Sie eine reguläre Definition an, die die Gleitpunktkonstanten der Sprache `C` beschreibt:

> **A.2.5.3 Gleitpunktkonstanten**
> [*Kernighan/Ritchie: Programmieren in C (2.Auflage)*]
>
> Eine Gleitpunktkonstante besteht aus einem ganzzahligen Teil, einem Dezimalpunkt, einem Dezimalbruch, dem Zeichen `e` oder `E`, einem ganzzahligen Exponenten mit optionalem Vorzeichen und einem optionalen Typ–Suffix, nämlich einem der Buchstaben `f`, `F`, `l` oder `L`. Ganzzahliger Teil und Dezimalbruch sind Ziffernfolgen. Entweder der Dezimalpunkt oder der Exponent beginnend mit `e`/`E` kann fehlen (aber nicht beide). Der Typ der Konstanten wird durch das Suffix bestimmt; `F` oder `f` machen sie zu `float`, die Suffixe `L` oder `l` machen sie zu `long double`; anderfalls hat sie den Typ `double`.

Aufgabe 11
In der Übung wurde die Funktion `foldr` definiert. Eine ähnliche Funktion `foldl` ist wie folgt in Haskell vordefiniert:

```
foldl :: (a -> b -> a) -> a -> [b] -> a
foldl f z []     = z
foldl f z (x:xs) = foldl f (f z x) xs
```

Veranschaulichen Sie sich einmal das Verhalten und den Typ der Funktionen `foldr` und `foldl`.

a) Welche Funktionen werden durch folgende Ausdrücke definiert (umgangssprachlich)?

- `foldr (:) []`
- `foldl (:) []`
- `foldl (-) 1`
- `foldr (-) 1`

b) Definieren Sie die Funktion `length` unter Verwendung der `fold`-Funktionen. Begründen Sie, weshalb Sie `foldr` bzw. `foldl` verwenden. Ist es auch möglich die duale `fold`-Funktion zu verwenden?

c) Definieren Sie die Funktion `reverse` unter Verwendung der `fold`-Funktionen. Ist eine der beiden Lösungen effizienter?