PHILIPPS-UNIVERSITÄT MARBURG
Fachbereich Mathematik und Informatik
Prof. Dr. R. Loogen J.Berthold

D-35032 Marburg
Hans Meerwein Straße
Lahnberge

10.November 2003

Übungen zur „Theorie des Compilerbaus“, WS 2003/04

Nr. 3 , Abgabe und Besprechung: 17. November in der Übung

Mündliche Aufgaben

3 .1 **lm-Zerlegung**
Beweisen oder widerlegen Sie:

> Seien $\alpha_1, \ldots, \alpha_n \in \mathrm{RA}(\Sigma)$ reguläre Ausdrücke und $w \in \Sigma^*$ ein Wort. Wenn es eine Zerlegung von w bzgl. $(\alpha_1, \ldots, \alpha_n)$ gibt, so gibt es auch eine lm-Zerlegung von w bzgl. $(\alpha_1, \ldots, \alpha_n)$.

3 .2 **Linearer Scan-Aufwand mit *Maximal-Munch***
In der Vorlesung wurde die sog. *Maximal-Munch*-Methode von Thomas Reps erwähnt, mit deren Hilfe ein Scanner in linearer Zeit arbeiten kann:

T. Reps: *Maximal-Munch Tokenization in Linear Time.* In: ACM TOPLAS 20(1998), 2,259ff.

Lesen Sie die genannte Veröffentlichung[1] und erklären Sie das beschriebene Verfahren an einem Beispiel.

Schriftliche Aufgaben

3 .3 **Römische Zahlen als reguläre Sprache** 6 Punkte
In dieser Aufgabe sollen römische Zahlen mit Hilfe regulärer Ausdrücke spezifiziert und bewertet werden.

Zusätzlich zu den Regeln aus Aufgabe 1.3 sollen in dieser Aufgabe Zahlen wie `VC` (Abziehen eines Zeichens mit Wert $5 \cdot 10^k$) verboten sein, ebenso Zahlen wie z.B. `XML` (nach der Subtraktion folgen Zeichen, deren Wert höher als das subtrahierte Zeichen ist).

(a) Geben Sie eine Folge von regulären Definitionen für römische Zahlen an (erweiterte Syntax zugelassen).

(b) Erstellen Sie daraus (mit beliebiger Methode) einen DFA, der korrekte römische Zahlen erkennt.

(c) Der DFA kann auch den Wert der gelesenen Zahl bestimmen. Versehen Sie dazu die Zustandsübergänge mit Werten, welche zum Wert der gelesenen Zahl summiert werden können.

3 .4 Umwandlung regulärer Ausdrücke zu NFAs in Haskell

6 Punkte

Reguläre Ausdrücke können in Haskell durch folgende Definitionen erfaßt werden:

Haskell Code

```
infixl 4 :|    -- Auswahl
infixr 5 :%    -- Sequenz
data RExp = Lam          -- Lambda
          | Ch Set       -- Buchstabe(n)
          | RExp :% RExp -- Sequenz
          | RExp :| RExp -- Auswahl
          | Star RExp    -- Stern
```

Die ersten beiden Zeilen definieren Assoziativität und Präzedenz der beiden Infix-Konstruktoren zur Darstellung von Sequenz und Auswahl. Mengen von Zeichen werden über ein Prädikat `Set` angegeben, wobei der Grundbereich die druckbaren ASCII-Zeichen seien. Als Beispiel:

```
digit::Set
digit x = x `elem` ['0'..'9']
```

Haskell Code

```
type Set = Char -> Bool -- Angabe einer Zeichenmenge über ein Prädikat

bereich :: Sigma
bereich = ['\33'..'\127'] -- Grundbereich: druckbare ASCII-Zeichen
```

(a) Definieren Sie eine Funktion `alphabet :: RExp -> Sigma`, welche zu einem regulären Ausdruck das zugrunde liegende Alphabet bestimmt.

(b) Aus einem regulären Ausdruck soll ein `NFA Int` induktiv konstruiert werden. Dazu ist es notwendig, an mehreren Stellen der Implementierung die Zustandsmenge eines NFA durch eine Verschiebung des Indexbereiches zu verändern, um NFAs zu vereinen. Die Zustände sind hier ganze Zahlen, so dass eine Indexverschiebung um n Stellen z.B. folgendermaßen aussieht:

```
[1, 2, 3] -> [1+n, 2+n, 3+n] = map (+n) [1, 2, 3]
```

Implementieren Sie eine Funktion

```
shift :: Int -> NFA' Int -> NFA' Int
type NFA'  state = ([state], DeltaNFA state, state, [state])
```

Der Aufruf `shift n nfa` soll dann den entsprechend transformierten Automaten zurückliefern. Das Alphabet soll über `alphabet` erkannt werden.

(c) Implementieren Sie eine Funktion

```
r2n :: RExp -> NFA Int
```

die induktiv über den Satz von Kleene aus einem regulären Ausdruck einen entsprechenden NFA konstruiert.

[1] Die Zeitschriften von ACM sind vom Fachbereich abonniert; Sie finden einen entsprechenden Link von der Homepage des Fachbereich 12 aus.