Prof. Dr. R. Loogen, M. Dieterle
Fachbereich Mathematik und Informatik
Hans-Meerwein-Straße
D-35032 Marburg

21. Oktober 2009

Philipps Universität Marburg

Übungen zu „Grundlagen des Compilerbau“, Winter 2009/10

Nr. 2, Abgabe der Aufgaben: 28. Oktober 2009 vor der Vorlesung

---

Aufgaben

2.1 Lexikalische Analyse Über dem Alphabet $\Sigma = \{A, B\}$ sind durch die folgenden regulären Ausdrücke drei Symbolklassen mit den angegebenen Prioritäten definiert: 5 Punkte

$$1: ABBA \qquad 2: A(B^+) \qquad 3: B^+A$$

(a) Erstellen Sie (ohne Potenzmengenkonstruktion) für jeden regulären Ausdruck einen DFA und konstruieren Sie daraus den Produktautomaten. Es kann davon ausgegangen werden, dass alle nicht gezeichneten Zustandsübergänge in den Senkenzustand führen. / 2

(b) Geben Sie die First-Longest-Match (flm) Zerlegung für die Worte / 2

$$ABBBA \qquad ABBBABBA \qquad ABBABBABBA$$

an. Begründen Sie ihre Antwort.

(c) Beweisen oder widerlegen Sie: Eine flm-Zerlegung existiert genau dann, wenn eine zulässige Zerlegung existiert. / 1

Bitte wenden!

2.2 Haskell-NFA

7 Punkte

Gegeben seien die folgenden Grunddefinitionen zur Behandlung von (nichtdeterministischen) endlichen Automaten in Haskell:

```
type Sigma       = [Char]
type DeltaNFA state = state → Maybe Char → [state]
data NFA state = NFA {states   :: [state],
                      alphabet :: Sigma,
                      trans    :: DeltaNFA state,
                      start    :: state,
                      accepts  :: [state]}
```

Der **Maybe**-Typ modelliert dabei die ε-Übergänge der Transitionsfunktion.[1] Mengen von Zuständen werden als Listen modelliert.[2]

(a) Konstruieren Sie (zunächst auf Papier) einen NFA für simple Bezeichner, die durch folgenden regulären Ausdruck definiert sind: / 1

$$(a \mid b)(a \mid b \mid 0 \mid 1)^*$$

Die Konstruktion soll formal **nach dem Satz von Kleene** geschehen und keine Vereinfachungen enthalten.

(b) Definieren Sie einen Haskell-NFA mit **Int**-Zuständen `bezNFA :: NFA Int`, der Ihrem Automaten aus (a) entspricht. / 2

Benutzung des Haskell-NFA

(c) Schreiben Sie eine Funktion `epsCl :: Eq st ⇒ DeltaNFA st → [st] → [st]`, welche die ε-Hülle einer Menge von Zuständen bestimmt. / 2

(d) Schreiben Sie nun eine Funktion / 2

```
checkWord :: Eq st ⇒ (NFA st) → String → Bool
```

die zu einem gegebenen NFA und einem Wort überprüft, ob das Wort von dem Automaten erkannt wird.

Hinweis: Im Wesentlichen ist die Funktion $\overline{\delta}$ für NFAs zu programmieren. Hilfreich sind die Funktionen höherer Ordnung **map**, **foldl** und **iterate** und "list-comprehensions".

---

1) Der Haskell-Typ **Maybe** a zu einem beliebigen Typ a besitzt die Konstruktoren **Nothing** und **Just**, wobei **Just** noch ein Element enthält. Mit Pattern Matching lassen sich die Fälle unterscheiden:

```
f Nothing  = "Keine Eingabe"
f (Just x) = "Eingabe " ++ show x
```

2) Das Modul **List** enthält hilfreiche Funktionen zur Mengendarstellung über Listen, etwa:

```
elem:: Eq a ⇒ a → [a] → Bool --"ist Element von"
concat:: [[a]] → [a]         --konkateniert Listen zu einer einzigen Liste
nub:: [a] → [a]              -- entfernt Duplikate aus einer Liste
union,intersect:: Eq a ⇒ [a] → [a] → [a]
                             -- Vereinigung und Schnitt von Listen
```

siehe auch: *http://www.haskell.org/onlinelibrary*