Grundlagen der praktischen Informatik
Dr. Stefan Siemer
M.Sc. Timo Specht

Sommersemester 2025
Georg-August-Universität Göttingen
Institut für Informatik
Lehrstuhl für Theoretische Informatik

Übungsblatt 4 (100 Punkte)

Reguläre Sprachen & Parser

Abgabe bis Di, 20. Mai, 14 Uhr.

Für das erfolgreiche absolvieren der Übungen, müssen Sie sich einmalig in FlexNow zum Übungsmodul **B.Inf.1102.Ue: Grundlagen der Praktischen Informatik - Übung** anmelden. Melden Sie sich **rechtzeitig** in **FlexNow** an.

Rechnerübung: Hilfe zu den praktischen Übungen können Sie in den Rechnerübungen bekommen. Details dazu werden noch bekannt gegeben.

Für die Prüfungszulassung benötigen Sie auf 8 verschiedenen Übungsblättern im theoretischen und praktischen Teil jeweils 40%, also jeweils 20 Punkte.

Theoretischer Teil

Hinweise zur Abgabe der Lösungen:

Die Lösungen der Aufgaben werden in geeigneter Form in der Stud.IP-Veranstaltung der Vorlesung über das Vips-Modul hochgeladen. Sie können Ihre Bearbeitungen gerne mit LaTeX formatieren, es ist aber auch der Upload von Text und Bilddateien in gängigen Formaten möglich.

Sollten Sie LaTeX nutzen wollen, gibt es hierfür Hinweise in Form eines Videos in Stud.IP und Sie können unser LaTeX-Template für Ihre Abgaben nutzen.

Die Aufgaben können in Gruppen von 2 bis 3 Personen bearbeitet und abgegeben werden. Wenn Sie in einer Gruppe abgeben möchten, achten Sie darauf, dass Sie sich vor der Abgabe im Vips-Modul der Vorlesung in einer Gruppe zusammenschließen.

Aufgabe 1 (15 Punkte)

Sind folgende Sprachen regulär? Wenn die Sprache regulär ist, beschreiben Sie oder geben Sie einen konkreten deterministischen endlichen Automaten an, der die Sprache erkennt.

1. Sei L die Sprache aller Wörter $w \in \{0, 1, \ldots, 9\}^*$ für die gilt, dass die Ziffernfolge w, interpretiert als ganze Zahl, ohne Rest durch 2 teilbar ist. Besteht die Ziffernfolge w aus mehr als einer Ziffer, muss die erste Ziffer ungleich 0 sein. Das leere Wort ist nicht in der Sprache enthalten.

   Ist die Sprache L regulär?
   (5 Punkte)

   Hinweis

   Ein Zahl ist ohne Rest durch 2 teilbar, wenn ihre letzte Ziffer ein 0,2,4,6 oder 8 ist.

2. Sei L die Sprache aller Wörter $w \in \{0, 1, \ldots, 9\}^*$ für die gilt, dass die Ziffernfolge w, interpretiert als ganze Zahl, ohne Rest durch 3 teilbar ist. Besteht die Ziffernfolge w aus mehr als einer Ziffer, muss die erste Ziffer ungleich 0 sein. Das leere Wort ist nicht in der Sprache enthalten.

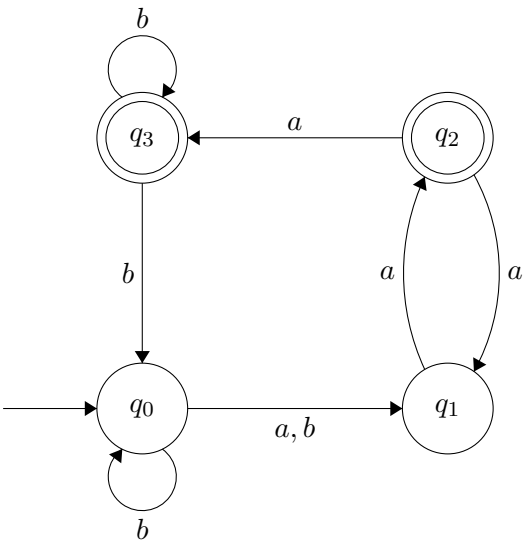
   Ist die Sprache L regulär?
   (10 Punkte)

   Hinweis

   Ein Zahl ist ohne Rest durch 3 teilbar, wenn die Quersumme ohne Rest durch 3 teilbar ist.

Aufgabe 2 (10 Punkte)

Geben Sie zu dem nachfolgend abgebildeten Zustandsgraphen eines nichtdeterministischen endlichen Automaten über dem Alphabet $\Sigma = \{a, b\}$ einen äquivalenten (vollständigen) deterministischen endlichen Automaten an. (10 Punkte)



Hinweis.

Bei einem (vollständigen) deterministischen endlichen Automaten ist für jeden Zustand für alle Zeichen des Alphabets ein nachfolgender Zustand definiert. Die Übergänge in den Fehlerzustand können dabei aber ausgelassen werden. Sie können den Automaten entweder durch einen Zustandsgraphen darstellen oder über eine Tabelle mit der Übergangsfunktion. Wollen Sie einen Zustandsgraphen erstellen, können sie folgende Seite verwenden: https://madebyevan.com/fsm/ Dort können Sie sich ebenfalls direkt LaTeX Code für den Automaten generieren lassen.

Aufgabe 3 (15 Punkte)

Beweisen Sie die folgenden Behauptungen.

1. Behauptung.

   Sind L_1, L_2 reguläre Sprachen, dann ist auch der Durchschnitt der beiden Sprachen regulär.

   $$L_1 \cap L_2 := \{w \mid w \in L_1 \text{ und } w \in L_2\}$$

   Seien $A_1 = (\Sigma_1, Q_1, q_1, F_1, \delta_1)$ und $A_2 = (\Sigma_2, Q_2, q_2, F_2, \delta_2)$ deterministische endliche Automaten, für die gilt A_1 akzeptiert L_1 und A_2 akzeptiert L_2.

   Zeigen Sie die Behauptung, indem Sie aus A_1 und A_2 einen deterministischen endlichen Automaten konstruieren, der $L_1 \cap L_2$ akzeptiert. (5 Punkte)

2. Behauptung.

   Ist L_1 eine reguläre Sprachen, dann ist auch ihr Komplement regulär.

   $$\overline{L_1} := \{w \mid w \notin L_1\}$$

   Sei $A_1 = (\Sigma_1, Q_1, q_1, F_1, \delta_1)$ ein deterministischer endlicher Automat, für den gilt A_1 akzeptiert L_1.

   Zeigen Sie die Behauptung, indem Sie aus A_1 einen deterministischen endlichen Automaten konstruieren, der $\overline{L_1}$ akzeptiert. (5 Punkte)

3. Behauptung.

   Sind L_1, L_2 reguläre Sprachen, dann ist auch die Vereinigung der beiden Sprachen regulär.

   $$L_1 \cup L_2 := \{w \mid w \in L_1 \text{ oder } w \in L_2\}$$

   Zeigen Sie die Behauptung, indem Sie die Ergebnisse aus den beiden vorherigen Aufgabenteilen nutzen. (5 Punkte)

Aufgabe 4 (10 Punkte)

Gegeben sei folgende Sprache über dem Alphabet $\Sigma = \{0, 1\}$.

$$L = \{w \in \Sigma^* \mid w \text{ enthält weder das Teilwort 00 noch das Teilwort 11}\}$$

1. Geben Sie die sieben kürzesten Wörter aus L an. (4 Punkte)
2. Geben Sie eine reguläre Grammatik G an, die L erzeugt. Benutzen Sie dabei höchstens 4 Nichterminale. (4 Punkte)
3. Zeigen Sie für die beiden längsten Wörter aus Aufgabenteil 1, jeweils durch Angabe einer Ableitung, das diese Wörter zur Sprache $L(G)$ gehören, die von der Grammatik aus Aufgabenteil 2. erzeugt wird. (2 Punkte)

Praktischer Teil

Hinweise zur Abgabe der Lösungen:

Die Prüfsumme wird in geeigneter Form in der Stud.IP-Veranstaltung der Vorlesung über das Vips-Modul hochgeladen. Jedes Aufgabenblatt hat die Aufgabe *Abgabe-Prüfsumme*. Dort hinterlegen Sie Ihre generierte Prüfsumme. Sie können zudem auch Ihren Quellcode dort hinterlegen.

Die Aufgaben können in Gruppen von 2 bis 3 Personen bearbeitet und abgegeben werden. Wenn Sie in einer Gruppe abgeben möchten, achten Sie darauf, dass Sie sich vor der Abgabe im Vips-Modul der Vorlesung in einer Gruppe zusammenschließen. Dabei ist zu beachten, dass die Gruppen für den theoretischen und praktischen Teil identisch sein müssen.

Aufgabe 5 (50 Punkte)

Parser

Zur Fehlerbehandlung kann die Funktion `error` aus `Prelude` verwendet werden, die als Argument eine Zeichenkette erwartet. Der Aufruf von `error` stoppt die Abarbeitung und gibt die Zeichenkette aus.

Beispiel

```haskell
-- calculate roots of  x^2 + p x + q = 0
parabolaRootA :: Double -> Double -> (Double, Double)
parabolaRootA p q
    | y < 0     = error "no roots"
    | otherwise = (-x + sqrt y, -x - sqrt y)
    where
        x = p/2.0
        y = x*x - q
```

Gibt es Nullstellen werde diese zurückgeliefert, ansonsten wird die Funktion abgebrochen und eine entsprechende Meldung ausgegeben.

```
> parabolaRootA 2.0 1.0
(1.0,1.0)
> parabolaRootA 1.0 2.0
no roots
```

Der Nachteil dieser Fehlerbehandlung ist das Abbrechen der Funktion.

Eine andere Möglichkeit zur Fehlerbehandlung ist die Benutzung des Typs `Maybe`.

```
data Maybe a = Just a | Nothing
    deriving (Eq, Ord)
```

Wird `Data.Maybe` importiert stehen u.a. die Funktionen

```
isNothing :: Maybe a -> Bool
fromJust :: Maybe a -> a
```

zur Verfügung. Die Funktion `isNothing` liefert genau dann `True` zurück, wenn das Argument `Nothing` ist. Die Funktion `fromJust` extrahiert das ursprünglich Element aus dem `Just`, ist das Argument `Nothing` kommt es zu einem Fehler.

Beispiel

```
-- calculate roots of  x^2 + p x + q = 0
parabolaRootB :: Double -> Double -> Maybe (Double, Double)
parabolaRootB p q
    | y < 0     = Nothing
    | otherwise = Just (-x + sqrt y, -x - sqrt y)
    where
        x = p/2.0
        y = x*x - q
```

Gibt es Nullstellen werde diese in einem `Just` zurückgeliefert, ansonsten wird `Nothing` zurückgeliefert.

```
> import Data.Maybe
> parabolaRootB 2.0 1.0
Just(-1.0,-1.0)
> parabolaRootB 1.0 2.0
Nothing
> fromJust (parabolaRootB 2.0 1.0)
(-1.0,-1.0)
> isNothing (parabolaRootB 1.0 2.0)
True
```

Der Vorteil dieser Methode ist, dass mit dem Rückgabewert der Funktion weitergearbeitet werden kann.

1. Programmieren Sie, nach dem Beispiel der Vorlesung für den rekursiven Abstieg, einen LL(1)-Parser für die folgende Grammatik.

   G = {N,T,P,S}

   - N = {`id_list`, `id_list_tail`}
   - T = {`"id"`, `","`, `";"`, `"$$"`}
   - S = `id_list`
   - P = { `id_list` → `"id" id_list_tail`
     `id_list_tail` → `"," "id" id_list_tail`
     `id_list_tail` → `";" "$$"` }

   Programmieren Sie mindestens folgende Funktionen, wobei das `[String]`-Argument die aktuell noch nicht abgearbeitete Eingabe repräsentiert.

```haskell
match :: String -> [String] -> [String]
id_list_tail :: [String] -> [String]
id_list :: [String] -> [String]
```

   Die Eingabe wird komplett abgearbeitete, wenn sie von der Grammatik erzeugt werden kann, d.h. die Funktion `id_list` liefert in diesem Fall eine leere Liste zurück. Behandeln Sie Fehler mit der Funktion `error`.
   (20 Punkte)

   Beispiel

```
> id_list ["id", ",", "id", ";", "$$"]
[]
> id_list ["id", "$$"]
error message
```

2. Programmieren Sie, nach dem Beispiel der Vorlesung für den rekursiven Abstieg, einen LL(1)-Parser für die folgende Grammatik.

   G = {N,T,P,S}

   - N = {`prog`, `expr`, `term`, `ttail`, `factor`, `ftail`}
   - T = {`'+'`, `'*'`, `'c'`, `'$'`}
   - S = `prog`
   - P = { `prog` → `expr '$'`
     `expr` → `term ttail`
     `term` → `factor ftail`
     `ttail` → `'+' term ttail` | ϵ
     `factor` → `'c'`
     `ftail` → `'*' factor ftail` | ϵ
     }

   Programmieren Sie für jedes Nichtterminal eine Funktion nach folgendem Vorbild, wobei das `Maybe String`-Argument die aktuell noch nicht abgearbeitete Eingabe repräsentiert, die auch `Nothing` (ein Fehler ist aufgetreten) sein kann.

```haskell
match :: Char -> Maybe String -> Maybe String
...
ttailA :: Maybe String -> Maybe String
expr :: Maybe String -> Maybe String
prog :: String -> Maybe String
```

Die Eingabe wird komplett abgearbeitete, wenn sie von der Grammatik erzeugt werden kann, d.h. die Funktion `prog` liefert in diesem Fall einen leeren `String` in einem `Just` zurück, kommt es zu einem Fehler wird `Nothing` zurückgeliefert. (30 Punkte)

Beispiel

```
> prog "c+c*c$"
Just""
> prog "c+c-c$"
Nothing
```