Grundlagen der Praktischen Informatik **Sommersemester 2023**
Dr. Henrik Brosenne Georg-August-Universität Göttingen
Institut für Informatik

Übungsblatt 04

E-Learning

Absolvieren Sie die Tests bis Di., 23.05., 8 Uhr

Die Tests sind in der Stud.IP-Veranstaltung *Grundlagen der Praktischen Informatik (Informatik II)* unter *Lernmodule* hinterlegt.

Sie können einen Test **nur einmal durchlaufen**. Sobald Sie einen Test starten steht Ihnen nur eine **begrenzte Zeit** zu Verfügung, um den Test zu bearbeiten.

Alle Punkte, die Sie beim Test erreichen, werden ihnen angerechnet.

ILIAS – 13 Punkte

Betriebssysteme - Prozess-Synchronisation, Speicherverwaltung

Absolvieren Sie die folgenden Tests.

- *GdPI 04 - Formale Sprachen - Repräsentationen von Automaten*
- *GdPI 04 - Formale Sprachen - Automat und Grammatik*
- *GdPI 04 - Formale Sprachen - Linkslineare Grammatik → rechtslineare Grammatik*

(13 Punkte)

Achtung

Zum ordnungsgemäßen Beenden eines Ilias-Test müssen Sie die Schaltfläche **Test beenden** betätigen.

Wenn Sie einen Ilias-Test einmal vollständig durchlaufen haben bekommen Sie auf die Seite *Testergebnisse*. Starten Sie den Test erneut aus Stud.IP, ist jetzt auch eine Schaltfläche *Testergebnisse anzeigen* vorhanden, die auf diese Seite führt.

Auf der Seite *Testergebnisse* können Sie sich unter *Übersicht der Testdurchläufe* zu jedem Testdurchlauf *Details anzeigen* lassen.

Prüfen Sie, insbesondere bei den *Markdown+AsciiMath*-Aufgaben, ob Ihre Lösung korrekt übermittelt wurde. Nur Lösungen, die hier angezeigt werden, können zur Korrektur in das Korrektursystem Grady übertragen werden.

Falls eine **Musterlösung** vorhanden ist, führt der Titel einer Aufgabe in der Auflistung der Aufgaben zur Musterlösung.

Hinweis

- Eine häufige Fehlerquellen ist das Schließen des Browser-Fensters vor **Test beenden**.
- Wenn Sie einen JavaScript Blocker einsetzen, sollten Sie für Ilias eine Ausnahme hinterlegen.

Übung

Abgabe bis Di., 23.05., 8 Uhr.

Allgemein

Ihre Lösungen werden nur korrigiert, wenn Sie einer Übungsgruppe angehören, d.h. Teilnehmer einer Stud.IP-Veranstaltung *GdPI - Übung - ⟨⟨Termin⟩⟩* sind.

Benutzen Sie Markdown mit AsciiMath um die Lösung der Aufgaben zu strukturieren und zu kodieren. Fassen Sie die **Lösungen aller Aufgaben in einer Datei** zusammen.

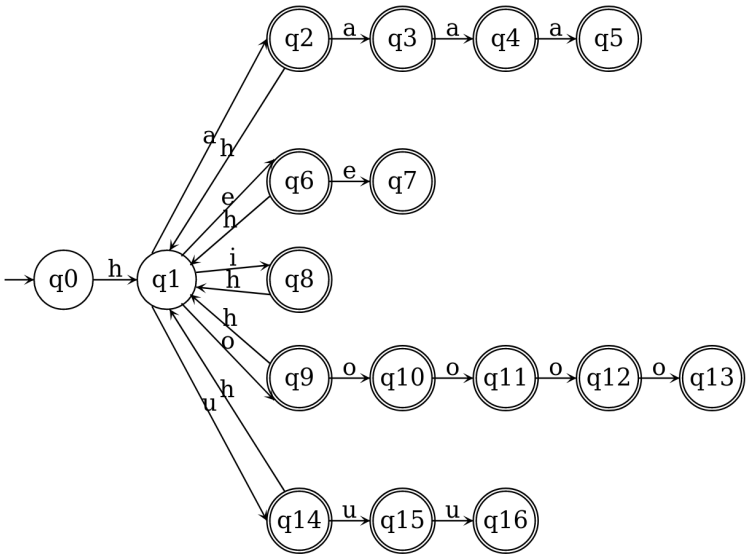
Formatieren Sie Ihre Abgabe so, dass **die einzelnen Aufgaben klar durch passende Überschriften voneinander** getrennt sind.

Geben Sie die Datei über das Lernmodul *GdPI 04 - Markdown+AsciiMath* ab.

Aufgabe 1 – 5 Punkte

Lachautomat

Der *Lachautomat* ist der folgende nichtdeterministische endlichen Automat über dem Alphabet der 26 Kleinbuchstaben.



Der folgende Text

```
heehihoohaOhoooahohishehuuuhaaaahuuehee
```

ist Teil der Niederschrift eines Gesprächs. Allerdings wurde bei der Aufzeichnung des Gesprächs im Hintergrund sehr viel gelacht, sodass dieses Gelächter im Text enthalten ist.

Der Lachautomat soll nun verwendet werden, um das Gelächter aus dem Text herauszufiltern. Dazu werden alle maximalen zusammenhängenden Zeichenketten, die der Automat akzeptiert, gelöscht. Mit dem Filtern wir am Anfang des Textes begonnen.

Geben Sie das so herausgefilterte Wort an.
(5 Punkte)

Aufgabe 2 – 12 Punkte

Reguläre Sprachen

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?
(12 Punkte)

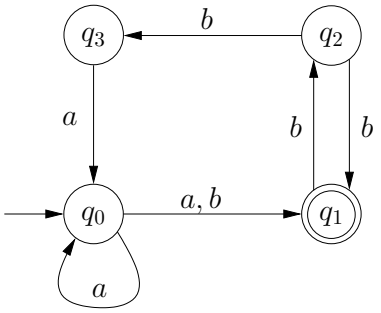
Hinweis

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

Aufgabe 3 – 20 Punkte

Äquivalente Automaten

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



Verwenden Sie dazu folgende Vorlage, zu finden als `dea.md` in der Stud.IP-Veranstaltung *Grundlagen der Praktischen Informatik (Informatik II)* unter *Übung→uebung04-data*.

```
## Deterministischer endlicher Automat (DEA)

$A = (Sigma, Q, s, F, delta)$

- Alphabet $Sigma = {a,b}$
- Zustände $Q = {{q_0}, ... }$
- Startzustand $s = {q_0}$
- Akzeptierende Zustände $F = { ... }$
- Überführungsfunktion $delta : Q xx Sigma -> Q$

### Überführungsfunktion

| $p in Q$          | $delta_p(a)$ | $delta_p(b)$ |
| :---              | :---         | :---         |
| ${q_0}$           | ${q_0, q_1}$ | ${q_1}$      |
| ${q_1}$           |              |              |
```

```
| ${q_2}$           |               |               |
| ${q_3}$           |               |               |
| ${q_0, q_1}$      |               |               |
| ${q_0, q_2}$      |               |               |
| ${q_0, q_3}$      |               |               |
| ${q_1, q_2}$      |               |               |
| ${q_1, q_3}$      |               |               |
| ${q_2, q_3}$      |               |               |
| ${q_0, q_1, q_2}$ |               |               |
| ${q_0, q_2, q_3}$ |               |               |
| ${q_1, q_2, q_3}$ |               |               |
| $0/$              |               |               |
```

Ergänzen Sie Zustände Q und akzeptierende Zustände F. Löschen Sie in der Tabelle alle Zeilen, deren Eintrag in der $p \in Q$ Spalte nicht zur Zustandsmenge gehört, vervollständigen Sie die übrigen Zeilen.

Hinweis.

Bei einem (vollständigen) deterministischen endlichen Automaten ist für jeden Zustand für alle Zeichen des Alphabets ein nachfolgender Zustand definiert.

Aufgabe 4 – 25 Punkte

Operationen auf regulären Sprachen

Behauptung.

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

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

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 deterministische endliche Automaten konstruieren, der $L_1 \cap L_2$ akzeptiert.
(25 Punkte)

Praktische Übung 04+05

Abgabe der Prüfsumme bis Di., 30.05., 8 Uhr
Testat Mo., 05.06. bis Mi., 14.06.

Hilfe zum Bearbeiten der praktischen Übungen können Sie grundsätzlich jeden Tag in den Rechnerübungen bekommen.

Am **Fr., 26.05.**, 8-10 Uhr in Präsenz (3 Tutor*innen) und 18-20 Uhr in Präsenz (1 Tutor*in) finden **keine Testate** statt. Diese Rechnerübungen sind ausschließlich **für Fragen** reserviert.

Abgabe der Prüfsumme

- Siehe vorherige Übungen.
- Übermitteln Sie die Prüfsumme mit dem Test *GdPI 04+05 - Testat*.

Aufgabe 1 – 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

```
-- 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.

```
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.

```
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
```