Prof. Javier Esparza
Philipp Czerner, Martin Helfrich

Technische Universität München
Lehrstuhl für Theoretische Informatik

Einführung in die Theoretische Informatik

Sommersemester 2021 – Hausaufgabenblatt 7

- Diese Woche werden alle Aufgaben korrigiert.
- Wenn Sie einen Beweis aufstellen, von dem Sie wissen, das einzelne Schritte problematisch oder unvollständig sind, merken Sie dies bitte in Ihrer Lösung an, damit wir das bei der Korrektur positiv berücksichtigen können.

Aufgabe H7.1. (*AT*) 0.5+0.5+0.5+0.5+1+1 Punkte

Diese Hausaufgabe wird mit Automata Tutor bearbeitet und abgegeben. Falls Sie es noch nicht bereits gemacht haben, folgen Sie den Schritten in Ü1.2, um ein Konto zu erstellen. Achten Sie darauf, dass Sie sich, wie dort beschrieben, mit Ihrer TUM-Kennung anmelden. Ansonsten können wir Ihnen die Punkte nicht gutschreiben.

Bearbeiten Sie die Hausaufgaben H7.1 (a–f). **Achtung:** Während Sie für die Aufgaben aus dem Übungsblatt beliebig viele Versuche hatten, haben Sie für jede Hausaufgabe nur 5 Versuche. Sie bekommen nur dann einen Punkt, wenn Sie die Aufgabe nach 5 Versuchen vollständig (also mit 10/10 Punkten) gelöst haben. Bei den *PDA construction* Aufgaben darf ihr konstruierter PDA nicht zu viele Zustände haben (siehe Aufgabenstellung). Wenn Sie einen ε-Übergang angeben wollen, geben Sie statt ε bitte `E` ein (siehe Hinweisbox über Canvas). Die Simulation bei PDAs ist deaktiviert. Bitte wundern Sie sich nicht, dass bei einem Klick auf `Start Simulation` nichts passiert.

Lösungsskizze.

(a)

$_{1,5}$									
$_{1,4}$	R	$_{2,5}$							
$_{1,3}$	S, R	$_{2,4}$	S	$_{3,5}$					
$_{1,2}$	S, A	$_{2,3}$	R	$_{3,4}$	S, R, B	$_{4,5}$			
$_{1,1}$	S, A, F	$_{2,2}$	S, A, F	$_{3,3}$	S, B, G	$_{4,4}$	S, B, G	$_{5,5}$	S, A, F
	x		a		x		b		x

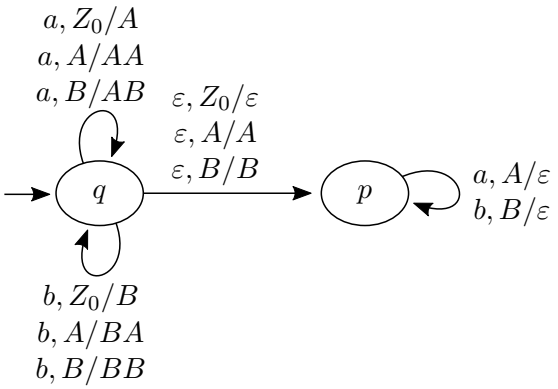
(b)

$_{1,5}$	S, T, H								
$_{1,4}$	S, T	$_{2,5}$	S, T, H						
$_{1,3}$	S	$_{2,4}$	S, T	$_{3,5}$	S				
$_{1,2}$		$_{2,3}$	S	$_{3,4}$	S	$_{4,5}$	S, H		
$_{1,1}$	L	$_{2,2}$	H	$_{3,3}$	L	$_{4,4}$	S, T, L	$_{5,5}$	L
	z		s		z		w		z

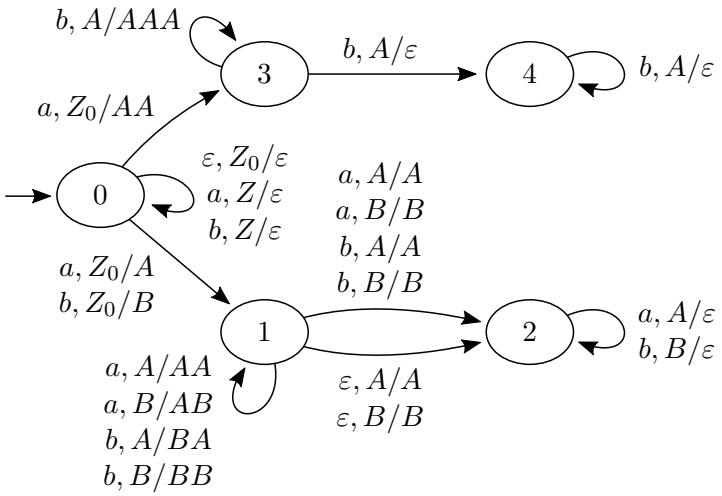
(c) $\epsilon, aabccb, acbccb \in L$ und $b, bb, abb \notin L$

(d) $\epsilon, aaabbaaa, ababbaba \in L$ und $bbb, aab, aaabaaa \notin L$

(e)



(f)



Aufgabe H7.2. (*Würze ∈ Kürze*) 2+2 Punkte

Der kleine Theodor muss morgen in der Schule einen Grammatiktest schreiben. Eigentlich sollte er lernen, aber er würde viel lieber den Blümchen beim Wachsen zuschauen. Die Grammatik hat er sich auch schon aufgeschrieben, aber leider passt sie nicht auf seinen Spickzettel. Können Sie Ihre moralischen Bedenken überwinden und ihn dabei unterstützen?

Die Grammatik G sei über die folgenden Produktionen gegeben:

$$
\begin{array}{ll}
S \to SS \mid AD \mid DB \mid T & E \to aABb \mid bBAa \mid EabU \\
A \to aT \mid aaD & T \to S \mid ETb \mid aAU \mid \varepsilon \\
B \to Ub \mid BB & U \to WDW \mid aEb \mid aU \\
C \to aV \mid \varepsilon & W \to aB \mid bAUb \mid bWa \\
D \to Sb \mid b & V \to aSb \mid ab
\end{array}
$$

(a) Eliminieren Sie alle unnützen Symbole aus G mit den aus der Vorlesung bekannten Verfahren. Geben Sie ihren Rechenweg an.

(b) Leider ist G noch nicht klein genug. Geben Sie eine Grammatik G' mit $L(G') = L(G)$ an, die höchstens zwei Produktionen enthält. Beschreiben Sie ihr Vorgehen.

Lösungsskizze. **(a)** Zuerst berechnen wir die Menge der erzeugenden Symbole. Jede Zeile

entspricht hierbei einem Schritt der Induktion aus Satz 4.37.

$$\begin{aligned}&\{a, b\}\\ &\cup \{C, D, T, V\}\\ &\cup \{A, S\}\\ &\cup \emptyset\end{aligned}$$

Hier ist die Induktion (bzw. der Fixpunktalgorithmus) also abgeschlossen, und wir erhalten folgende Grammatik:

$$\begin{aligned} S &\to SS \mid AD \mid T & D &\to Sb \mid b\\ A &\to aT \mid aaD & T &\to S \mid \varepsilon\\ C &\to aV \mid \varepsilon & V &\to aSb \mid ab \end{aligned}$$

Nun bestimmen wir die Symbole, die erreichbar sind, induktiv nach Satz 4.40.

$$\begin{aligned}&\{S\}\\ &\cup \{A, D, T\}\\ &\cup \emptyset\end{aligned}$$

Schließlich ergibt sich die folgende Grammatik.

$$\begin{aligned} S &\to SS \mid AD \mid T & D &\to Sb \mid b\\ A &\to aT \mid aaD & T &\to S \mid \varepsilon \end{aligned}$$

(b) Zuerst eliminieren wir T, da offensichtlich T und S äquivalent sind.

$$\begin{aligned} S &\to SS \mid AD \mid \varepsilon & D &\to Sb \mid b\\ A &\to aS \mid aaD \mid a \end{aligned}$$

Die Produktionen von D lassen sich einsetzen.

$$\begin{aligned} S &\to SS \mid ASb \mid Ab \mid \varepsilon\\ A &\to aS \mid aaSb \mid aab \mid a \end{aligned}$$

Ebenso die von A.

$$S \to SS \mid aSSb \mid aaSbSb \mid aabSb \mid aSb \mid aaSbb \mid aabb \mid ab \mid \varepsilon$$

Die Produktionen $S \to aSSb \mid aaSbSb \mid aabSb \mid aaSbb \mid aabb \mid ab$ lassen sich nun entfernen, da man jeweils die rechte Seite auch in folgender Grammatik ableiten kann.

$$S \to SS \mid aSb \mid \varepsilon$$

Diese Grammatik erzeugt genau die balancierten Klammerwörter (mit a als öffnende und b als schließende Klammer), wie aus der Vorlesung (Beispiel 4.10) bekannt. Dafür können wir aber auch eine kleinere Grammatik angeben:

$$S \to aSbS \mid \varepsilon$$

Aufgabe H7.3. (*Hochstaplerei*) 1+3 Punkte

Dora ist wütend. Ihr Kindergartenrivalin, Eva Pirsalz, behauptet einen größeren Bausteinturm als Dora gebaut zu haben. Aber als Dora sich den Turm anschauen wollte, war er nicht mehr da – Eva behauptet, sie hat ihn schon wieder abgebaut. Dora will der Sache auf den Grund gehen, und besorgt sich eine Kopie von Evas Labortagebuch. Da muss für jeden Baustein jeweils vermerkt sein, wann der er aus der Bausteinbox **g**eholt wurde, wann er **v**erbaut wurde, und wann er wieder **z**urückgelegt wurde.

Die Einträge werden mit dem Alphabet $\Sigma := \{g, v, z\}$ notiert. In Doras Labortagebuch steht z.B. das Protokoll $ggvgvvzgvzzz$. Ein Baustein muss immer zuerst geholt, dann verbaut, und dann zurückgegeben werden. Das Protokoll $gvvgzz$ wäre also nicht gültig, da hier ein Stein verbaut wurde, bevor er geholt wurde. Ebenso wäre $ggvgvvzz$ nicht in Ordnung, da einer der Steine nicht zurückgegeben wurde, oder gz, da einer der Steine nicht verbaut wurde. Sei $L := \{w \in \Sigma^* : w \text{ ist gültiges Protokoll}\}$. Um Eva zu überführen, will Dora nun einen Kellerautomaten bauen, der jedes Protokoll in Evas Labortagebuch überprüft.

(a) Zeigen Sie, dass L nicht kontextfrei ist und es somit keinen Kellerautomaten für L gibt.

Dora ist enttäuscht, hat aber noch eine Idee: Für Ihre Zwecke wäre es genauso gut, einen Kellerautomaten zu haben, der die *ungültigen* Protokolle akzeptiert.

(b) Konstruieren Sie einen Kellerautomaten für $\overline{L}$ und beschreiben Sie die Idee hinter Ihrer Konstruktion.

Hinweis: Bitte beachten Sie die Anmerkungen zur Notation von Kellerautomaten auf Übungsblatt 7.

Lösungsskizze. **(a)** Angenommen, L wäre kontextfrei. Dann können wir das Pumping-Lemma für kontextfreie Sprachen anwenden.

Sei $n \in \mathbb{N}$ PL-Zahl. Wir wählen das Wort $r := g^n v^n z^n$. (Wir bezeichnen dies meist als z, das wäre hier aber ungünstig, da im Alphabet Σ. Also wählen wir r statt z für den Namen des Wortes, und f statt v für den zweiten Teil der Zerlegung.)

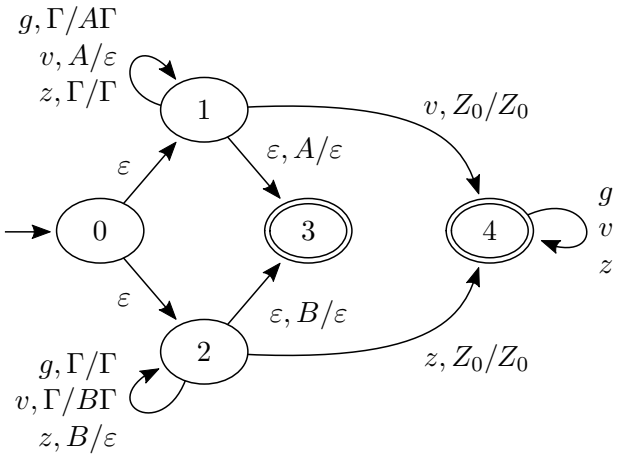
Sei $ufwxy$ eine Zerlegung unseres Wortes mit $|fwx| \leq n$ und $fx \neq \varepsilon$. Wir betrachten die folgenden Fälle:

- $|fx|_z = 0$: Wir pumpen also kein z. Wir pumpen ab, betrachten also das Wort $r' := uf^0wx^0y = uwy$. Da $fx \neq \varepsilon$, haben wir zumindest ein g oder ein f entfernt. Daher gilt $|r'|_g < |r'|_z$ oder $|r'|_v < |r'|_z$. Das heißt, ein Stein wurde zurückgegeben bevor er geholt oder verbaut wurde. Daher ist in beiden Fällen das Wort $r' \notin L$.
- $|fx|_z > 0$: Aufgrund der Form unseres Wortes und $|fwx| \leq n$ gilt also $|fwx|_g = 0$. Wir pumpen 2-mal, betrachten also $r' := uf^2wx^2y$. Es gilt $|r'|_g = n < n + |fx|_z = |r'|_z$. In r' wurde daher ein Stein zurückgegeben, bevor er geholt wurde, und $r' \notin L$.

(b) Um zu überprüfen, dass ein Wort in L enthalten ist, können wir folgendermaßen vorgehen: Wir gehen das Wort Zeichen für Zeichen durch, und merken uns stets die Anzahl der Steine die wir geholt, aber nicht verbaut haben, und die der Steine, die wir verbaut, aber nicht zurückgelegt haben. (Erstere Zahl nennen wir A, und letztere B.)

Keine dieser Zahlen darf negativ werden, und am Ende müssen beide 0 sein. Mit einem Kellerautomaten kann man dies nicht überprüfen, da er sich nur eine dieser Zahlen merken kann. Aber: Falls das Wort nicht in der Sprache ist, muss also einer dieser

Zähler negativ werden oder nicht auf 0 enden. Dies können wir überprüfen, indem wir den Nichtdeterminismus ausnutzen.



(Der PDA verwendet die auf Übungsblatt 8 eingeführte Kurznotation.) Zuerst entscheidet sich der Automat, ob er A oder B überprüft. Wir betrachten nun den Fall, dass der Automat A überprüft, das Vorgehen für B ist analog. Anfangs setzten wir den Zähler auf 1, indem er ein A auf den Stapel legt. Danach liest er das Wort ein. Jedes g legt ein weiteres A auf den Stapel, jedes v nimmt eines herunter. Es gibt zwei Möglichkeiten, wie das Wort inkorrekt sein könnte: (1) Der Zähler erreicht zwischendurch die Zahl 0 (es wurde also ein Stein verbaut, der nicht geholt wurde). (2) Der Zähler endet nicht bei 1 (es wurden nicht alle Steine verbaut).

Im Fall (1) geht der Automat zu Zustand 4 über, und kann dort den Rest des Wortes akzeptieren. (Umgekehrt kann man Zustand 4 auch nur in Fall (1) erreichen.) Bei (2) müssen am Ende des Wortes zwei A auf dem Stapel liegen (der Zähler ist also mindestens 2) – hier gehen wir nach 3, von wo aus keine weiteren Zeichen mehr eingelesen werden können. Auch hier gilt die Umkehrung: wir können nur nach 3 gehen, wenn wir am Ende des Wortes sind und mindestens zwei A auf dem Stapel liegen.

Bonusaufgabe H7.4. 2 Bonuspunkte

Implementieren Sie ein Programm, das beliebige theoLISP Programme ausführen kann (wie in Aufgabe H6.3 definiert). Führen Sie so das (korrigierte) Programm aus Aufgabe H6.4 aus und geben Sie das Ergebnis an.

Hinweise: Es ist sinnvoll, hierzu Ihre Lösung von Aufgabe H6.4 zu erweitern. Alternativ können Sie auch auf der Musterlösung aufbauen, diese finden sie hier, zusammen mit dem korrigierten Programm aus Aufgabe H6.4. Beachten Sie bitte auch die Beispielprogramme zu Aufgabe H6.3 (diese sind hier zu finden), für die die korrekten Ausgaben jeweils angegeben sind.

Wie immer: Beschreiben Sie bitte Ihren Ansatz in *natürlicher Sprache* und illustrieren die wesentlichen Schritte Ihrer Lösung mit geeigneten Codefragmenten. Sie können (müssen aber nicht), Ihrer Lösung Ihren vollständigen Programmcode beifügen, jedoch steht es den Korrektoren frei, diesen zu ignorieren. Beschränken Sie sich in Ihrer Implementierung auf grundlegende Funktionalitäten verbreiteter Programmiersprachen.

Es genügt zu beschreiben, wie Sie ihren Recogniser (oder den der Musterlösung) anpassen. Sie müssen nicht erneut beschreiben, wie dieser funktioniert.

Knobelaufgabe: Wenn Sie herausfinden, was das Programm tatsächlich ausrechnet (für beliebige Eingaben n), können Sie gerne eine E-Mail an Philipp Czerner schicken.

Lösungsskizze. Zuerst ändern wir unseren Recogniser, sodass er eine verschachtelte Liste mit dem Programminhalt zurückgibt. Für das Programm

```
((while x ()) (set x (add 42 7)))
```

erhalten wir z.B.

$$[[\texttt{'while'}, \texttt{'x'}, []], [\texttt{'set'}, \texttt{'x'}, [\texttt{'add'}, 42, 7]]]$$

Die Liste entspricht also exakt der S-Expression; Zahlen sind als Zahlen repräsentiert, und alles andere als String. In unserem Parser haben wir bereits für jede Variable eine Funktion, die diese Variable einliest – wir müssen die Funktion nun so anpassen, dass sie eine entsprechende Liste zurückgibt. Beispiel:

```
def Zahl():
    r = ''
    assert code[0].isdigit()
    while  code[0].isdigit(): r += code[0]; eat(code[0])
    return int(r)
```

Hier wird eine Zahl eingelesen und zu einem Integer konvertiert. Die meisten dieser Funktionen arbeiten rekursiv:

```
def Term():
    if eat_maybe('('): r = Arith(); eat(')'); return r
    elif code[0].isdigit(): return Zahl()
    elif code[0].isalpha(): return Var()
    else: assert False
def Arith():
    r1 = Op();   eat(' ')
    r2 = Term(); eat(' ')
    r3 = Term()
    return [r1, r2, r3]
def While():
    eat('while'); eat(' ')
    r2 = Term();  eat(' ')
    r3 = S()
    return ['while', r2, r3]
```

Das Vorgehen ist stets das gleiche, die anderen Funktionen sind deshalb hier nicht gezeigt. Nun können wir das Programm einlesen.

```
program = S()
```

Um das Programm dann auszuführen, gehen wir wieder rekursiv vor. Wir erstellen drei Funktionen: `run`, um eine Liste an Befehlen auszuführen; `run_single`, für einen einzelnen Befehl, und `calc`, um den Wert eines arithmetischen Ausdrucks zu ermitteln. Die Implementation von `run` ist leicht.

```
    def run(program):
        for i in program: run_single(i)
```

Um `calc` zu implementieren, müssen wir uns den aktuellen Wert aller Variablen merken. Hierzu verwenden wir ein globales Dictionary.

```
variables = {}
def calc(E):
    if   isinstance(E, int): return E
    elif isinstance(E, str): return variables.get(E, 0)
    elif E[0] == 'add': return calc(E[1]) + calc(E[2])
    elif E[0] == 'mul': return calc(E[1]) * calc(E[2])
    elif E[0] == 'sub': return calc(E[1]) - calc(E[2])
```

Beachten Sie, dass wir für Variablen, denen bisher kein Wert zugewiesen wurde, den Wert 0 verwenden. Schließlich implementieren wir `run_single`, unter Verwendung der beiden vorherigen Funktionen.

```
def run_single(P):
    if P[0] == 'if':
        if calc(P[1]) > 0: run(P[2])
        else:              run(P[3])
    elif P[0] == 'while':
        while calc(P[1]) > 0: run(P[2])
    elif P[0] == 'set':
        variables[P[1]] = calc(P[2])
```

Nun führen wir das Programm aus und geben das Ergebnis zurück.

```
    run(program)
    print(calc('result'))
```

Der Ergebnis ist 347. (Und tatsächlich wurde im Jahre 347 der spätere Kaiser Theodosius der Große geboren.)