Prof. Dr. Michael Jäger FB MNI

Lösungsvorschlag zur Klausur „Compilerbau“ vom 25. September 2018

Aufgabe 1 (2+2+2+2+2) *Punkte* ____ *von* **10**

a) Den Übersetzungsprozess kann man untergliedern in „Analyse“ und „Synthese“. Im Praktikum erfolgte eine Aufteilung der Compiler-Komponenten in „Hausübung 1“ und „Hausübung 2“. Entspricht „Hausübung 1“ der Analyse und „Hausübung 2“ der Synthese? (Begründung)

Nein, HÜ2 umfasst die Synthese, aber auch die semantische Analyse, die zur Analysephase gehört. Semantische Analyse und Synthese werden deshalb in eine Hausübung zusammengefasst, weil es sich jeweils um Algorithmen auf dem abstrakten Syntaxbaum handelt.

b) In welcher Hinsicht spielen endliche Automaten beim Compilerbau eine Rolle?

Aus regulären Ausdrücken für die Token-Syntax lassen sich systematisch bzw. automatisch endliche Automaten konstruieren, die in Verbindung mit einem DEA-Simulator zur lexikalischen Analyse genutzt werden können (z.B. jflex).

Ein Bottom-Up-Parser benutzt einen DEA, dessen Zustand den Fortschritt beim Aufbau einer rechten Regelseite im Parserstack wiedergibt. Der Zustand und das nächste Token bestimmen die nächste Aktion.

c) Nehmen Sie an, die Operatoreigenschaften für Subtraktion und Division wären nicht wie gewohnt, sondern wie folgt definiert:

- Subtraktionsoperator ($-$): hohe Präzedenz, linksassoziativ
- Divisionsoperator ($/$): niedrige Präzedenz, rechtsassoziativ

Welchen Wert hätte dann der folgende Ausdruck? `100/2-3/5-2-1/5/5`

`100/2-3/5-2-1/5/5 = 100/((2-3)/(((5-2)-1)/(5/5))) = 100/(-1/2) =` -200

d) Sei G=(T,N,P,S) eine kontextfreie Grammatik. Dabei ist P die Menge der Ableitungsregeln. Was ist eine Ableitungsregel? Geben Sie eine exakte Definition an.

Eine Ableitungsregel hat die Form

$\alpha \to w$, wobei $a \in N, w \in (N \cup T)^*$

α heißt linke Seite, w rechte Seite der Regel.

e) Warum berechnet ein Top-Down-Parser eine Linksableitung und ein Bottom-Up-Parser eine Rechtsableitung?

Jeder Parser verarbeitet die Eingabe von links nach rechts, damit er das Vorausschau-Token für seine Entscheidungen nutzen kann. Am Beispiel $S \to ABC$ sieht man, dass dazu die Regel für A zuerst bestimmt werden muss, was einer Linksableitung entspricht.

Der Bottom-Up-Parser berechnet Ableitungen aber in umgekehrter Reihenfolge beginnend mit dem letzten Ableitungsschritt. Je weiter hinten ein Ableitungsschritt innerhalb der Ableitung steht, desto früher wird er also berechnet. Da A bei einer Rechtsableitung zuletzt ersetzt wird, wird bei der Rückwärts-Konstruktion dieser Ableitung die Regel für A zuerst bestimmt. Der Parser geht dabei von links nach rechts durch die Eingabe.

Aufgabe 2 (3+4+3 Punkte)

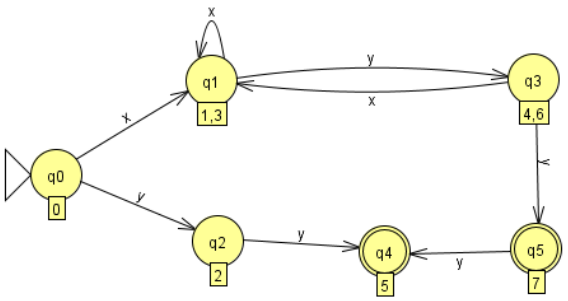
Punkte ____ *von* **10**

a) Beweisen Sie, dass die folgenden beiden regulären Audrücke nicht äquivalent sind.

$$r_1 = ((xy \mid yx \mid \varepsilon)(x^*y^*))^+, \quad r_2 = ((y \mid x)(xyx \mid \varepsilon))^+$$

r_1 und r_2 sind äquivalent, g.d.w. $L(r_1) = L(r_2)$. Da $\varepsilon \in L(r_1), \varepsilon \notin L(r_2)$ gilt $L(r_1) \neq L(r_2)$.
Hinweis: $L(r_1) = \{x, y\}^*, L(r_2) = \{x, y\}^+$

b) Geben Sie zu dem regulären Ausdruck $r_3 = (x \mid xy \mid \varepsilon)^+yy$ einen äquivalenten endlichen Automaten an. Die volle Punktzahl gibt es nur für einen deterministischen Automaten, Nichtdeterminismus führt zur Abwertung.



(mit JFLAP generiert)

c) Geben Sie zu ihrem Automaten aus (c) und folgenden Eingabewörten aus $L(r_3)$ jeweils eine akzeptierende Berechnung an:

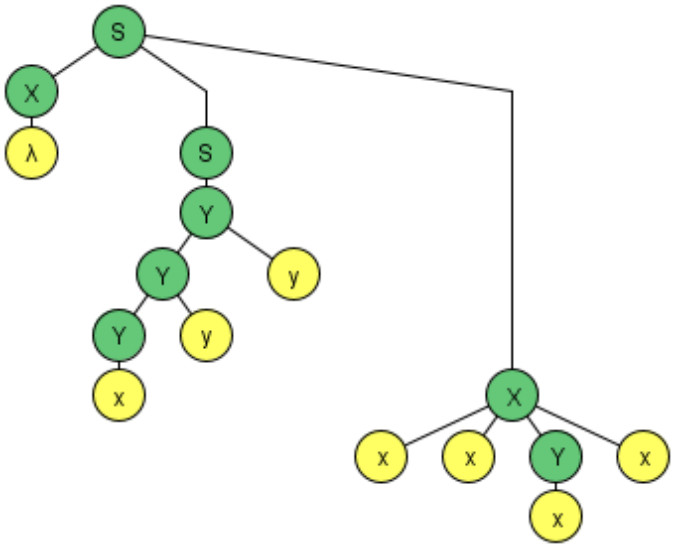
- yy $\quad q0 \overset{y}{\Rightarrow} q2 \overset{y}{\Rightarrow} q4$
- xyy $\quad q0 \overset{x}{\Rightarrow} q1 \overset{y}{\Rightarrow} q3 \overset{y}{\Rightarrow} q5$
- $xyyy$ $\quad q0 \overset{x}{\Rightarrow} q1 \overset{y}{\Rightarrow} q3 \overset{y}{\Rightarrow} q5 \overset{y}{\Rightarrow} q4$

Aufgabe 3 (2+2+2+2+2 Punkte)

Punkte ____ *von* **10**

Gegeben sei folgende kontextfreie Grammatik G: $S \to XSX \mid Y$, $X \to xxYx \mid \varepsilon$, $Y \to Yy \mid x$

a) Geben Sie den Ableitungsbaum und eine Linksableitung zu $xyyxxxx$ an.



Linksableitung: $S \Rightarrow XSX \Rightarrow SX \Rightarrow YX \Rightarrow YyX \Rightarrow YyyX \Rightarrow xyyX \Rightarrow xyyxxYx \Rightarrow xyyxxxx$

b) Beweisen Sie, dass die Grammatik mehrdeutig ist.

G ist mehrdeutig, g.d.w. es für ein Wort $w \in L(G)$ zwei Linksableitungen gibt. Dies ist für $w = x$ der Fall:

$S \Rightarrow Y \Rightarrow x$

$S \Rightarrow XSX \Rightarrow SX \Rightarrow YX \Rightarrow xX \Rightarrow x$

c) Bestimmen Sie $FIRST(S)$: $\{x\}$

d) Bestimmen Sie $FOLLOW(Y)$: $\{x, y, \$\}$

e) Was steht in der LL(1)-Parsertabelle in dem Eintrag zu X und x ? $X \to \varepsilon$ und $X \to xxYx$.

Punkte ____ *von* **10**

Aufgabe 4 (4+3+3 Punkte)

a) Bestimmen Sie zur nachfolgenden Grammatik G die LR(0)-Elemente und die Übergänge im zugehörigen DEA.

b) Geben Sie die SLR(1)-Parsertabelle dazu an.

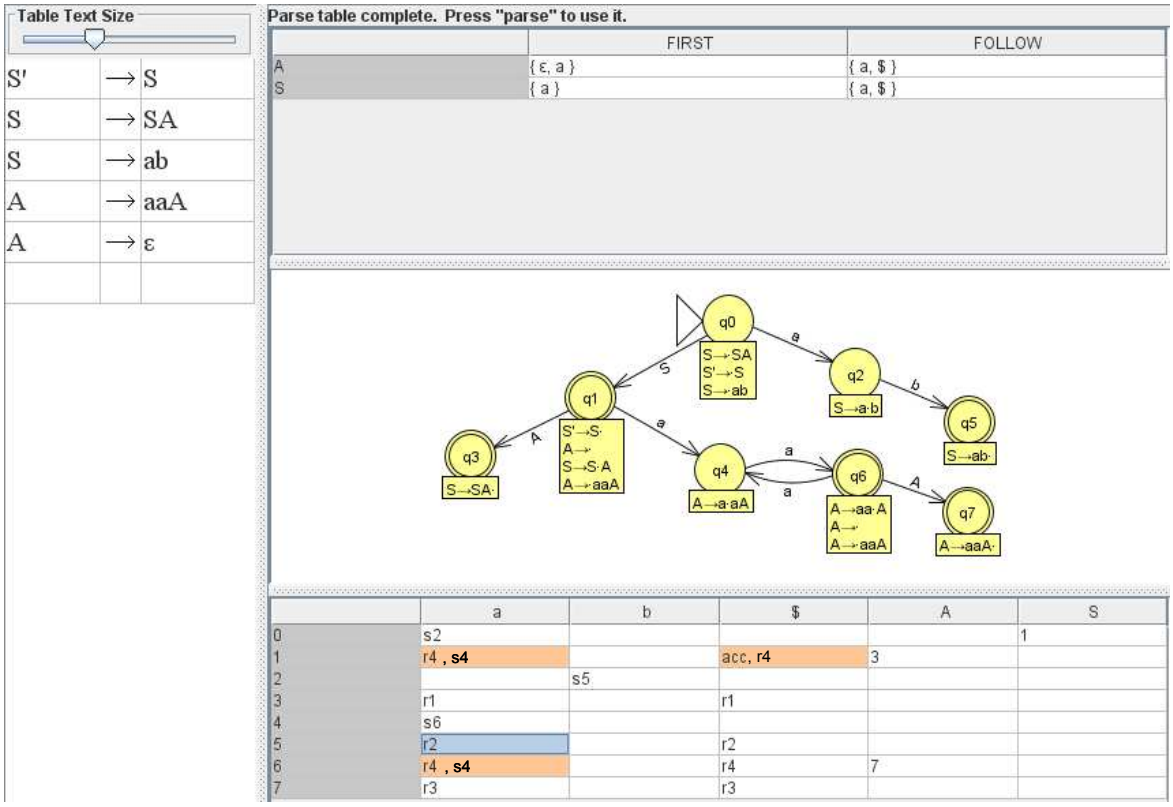
c) Geben Sie die Berechnung des SLR(1)-Parsers für die Eingabe $abaa$ an. Falls die Tabelle Shift/Reduce-Konflikte enthält, soll der Parser dabei immer die SHIFT-Aktion wählen.

(1) $S \to SA$
(2) $S \to ab$
(3) $A \to aaA$
(4) $A \to \varepsilon$

(4c)

Nr.	Stack	Resteingabe	Aktion
1	0	abaa	s2
2	0a2	baa	s5
3	0a2b5	aa	r2
4	0S1	aa	s4
5	0S1a4	a	s6
6	0S1a4a6		r4
7	0S1a4a6A7		r3
8	0S1A3		r1
9	0S1		accept

(a,b)



Aufgabe 5 (3 + 7 Punkte)

Punkte ____ von **10**

Betrachten Sie folgende SPL-Definitionen:

```
type vektor = array [10] of int;

proc p (i:int, ref v1:vektor) {
    var j: int;
    var v2:vektor;
    j := i+1;
    printi(v1[j]);
}
```

a) Bestimmen Sie das Frame-Layout für den Aktivierungsrahmen von p durch Ausfüllen jeweils einer Tabellenzeile pro Speicherplatz im Frame

Inhalt	Größe in Bytes	Offset zu FP	Offset zu SP
j	4	-4	52
v2	40	-44	12
FP alt	4	-48	8
RETURN alt	4	-52	4
Argument 1 für printi	4	-56	0

b) Bestimmen Sie den Puck-Assemblercode zu den beiden Anweisungen im Rumpf von p (ohne Prolog und Epilog). Wahlweise können Sie auch den ECO32-Assemblercode angeben. Die Prozedur *printi* erwartet einen Wertparameter vom Typ int.

Registernutzung:

Puck-VM: SP=$31, FP=$29, RET=$30, Register für Zwischenwerte: $8-$15
ECO32-VM: SP=$29, FP=$25, RET=$31, Register für Zwischenwerte: $8-$15

```
ADDC $8 $29 -4      ; $8  <-- &j
ADDC $9 $29 0       ; $9  <-- &i
LDW $9 $9           ; $9  <-- i
SETW $10 1          ; $10 <-- 1
ADD $9 $9 $10       ; $9  <-- i+1
STW $9 $8           ; j   <-- i+1

ADDC $8 $29 4       ; $8  <-- & Argument 2
LDW $8 $8           ; $8  <-- &v1  (Wert von Argument 2)
ADDC $9 $29 -4      ; $9  <-- &j
LDW $9 $9           ; $9  <-- j
SETW $10 10         ; $10 <-- 10 (Anzahl der vektor-Komponenten)
LTU $10 $9 $10      ; $10 <-- (0 <= j < 10)
BRF $10 indexError
SETW $10 4          ; $10 <-- 4 (Byteanzahl pro vektor-Komponente)
MULU $9 $9 $10      ; $9  <-- j*4 (Byte-Offset zu &v1)
ADD $8 $8 $9        ; $8  <-- &v1[j]  (&v1 + 4*j)
LDW $8 $8           ; $8  <-- v1[j]
ADDC $9 $31 0       ; $9  <-- & Argument 1 für Callee
STW $8 $9           ; Argument 1 für Callee <-- $8
CALL $30 printi     ; Aufruf printi
```