Prof. Dr. Michael Jäger FB MNI

Lösungsvorschlag zur Klausur „Compilerbau“ vom 19. September 2019

Punkte ____ *von* **10**

Aufgabe 1 (2+2+2+2+2)

a) Ein Compiler analysiert einen Quelltext. Diese Analyse wird wegen der Verwendung unterschiedlicher Analysetechniken in Teilaufgaben zerlegt, die durch separate Compilerkomponenten implementiert werden. Welche Teilaufgaben sind das?

- Lexikalische Analyse
- Syntaxanalyse
- Semantische Analyse

Anmerkung: Es war nach den Teilaufgaben der **Analyse** gefragt!

b) Ein Parsergenerator (z.B. cup oder bison) erzeugt aus einer Grammatik einen Parser. In welcher Hinsicht bietet er einen darüber hinaus gehenden Nutzen? Geben Sie dazu auch Beispiele an.

Die Grammatik lässt sich durch Attributierung der Symbole und semantische Aktionen so erweitern, dass während der Syntaxanalyse weitere syntaxorientierte Algorithmen spezifiziert werden können. Der Generator erzeugt dann einen Parser, der die Attribute auf dem Stack verwaltet und die semantischen Aktionen ausführt. Beispiele für syntaxorientierte Algorithmen: Aufbau eines abstrakten Syntaxbaums, semantische Analyse, Codeerzeugung.

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

- Additionsoperator ($+$): hohe Präzedenz, rechtsassoziativ
- Divisionsoperator ($/$): mittlere Präzedenz, linksassoziativ
- Multiplikationsoperator ($*$): niedrige Präzedenz, rechtsassoziativ

Welchen Wert hätte dann der folgende Ausdruck? `4+2/3*(8/1+1)*2+4`

Vollständig geklammerter Ausdruck: `((4+2)/3)*((8/(1+1))*(2+4))`
Wert: `2 *( 4 * 6 ) = 48`

d) Sei G=(T,N,P,S) eine kontextfreie Grammatik. Dabei ist P die Menge der Ableitungsregeln, aus denen sich Ableitungen bilden lassen. Eine Ableitung ist eine Folge von Ableitungsschritten. Was ist ein Ableitungsschritt? Geben Sie eine exakte Definition an.

Anmerkung: Die im Skript definierte „direkte Ableitbarkeit“ entspricht einem Ableitungsschritt, die „Ableitbarkeit“ einer Folge von Ableitungsschritten.

Ein Ableitungsschritt ist die **Anwendung einer Ableitungsregel** $x \to y$ auf eine Satzform uxv mit dem Ergebnis uyv. Dabei gilt: $u, x, y, v \in (T \cup N)^*, (x \to y) \in P$.

e) Welche Vorteile hat die Spezifikation der Token-Syntax durch reguläre Ausdrücke?

- Die formale Spezifikation durch reguläre Ausdrücke ist kurz, eindeutig und vollständig.
- Sie ermöglicht die automatische Generierung von endlichen Automaten und darauf aufbauenden Scanner-Implementierungen mittels Scannergeneratoren wie *flex* oder *jflex*.

Aufgabe 2 (3+3+3+1 Punkte)

Punkte ____ von **10**

a) Geben Sie zu folgender Bezeichnersyntax einen passenden regulären Ausdruck an:

Bezeichner beginnen mit einem Buchstaben oder einem Unterstrich. Die restlichen Zeichen müssen Buchstaben oder Ziffern sein. Wenn aber das erste Zeichen ein Unterstrich ist, dürfen bei den restlichen Zeichen zusätzlich $-Zeichen vorkommen.

```
[a-zA-Z][a-zA-Z0-9]*|_[a-zA-Z0-9$]*
```

Anmerkungen: Auch `[_a-zA-Z][a-zA-Z0-9]*|_[a-zA-Z0-9$]*` ist korrekt. Auch +-Operatoren (statt $*$) werden als korrekt gewertet, da nicht klar spezifiziert ist, ob es außer dem ersten Zeichen noch weitere Zeichen geben muss.

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

$$r_1 = ((\varepsilon \mid a \mid ab \mid ba)b)^*, \quad r_2 = ((a \mid b)^*b)^*$$

Lösungsansatz:

Gesucht ist ein Wort, dass zu einem der beiden Ausdrücke passt, zu dem anderen aber nicht. Bei eingehender Betrachtung der regulären Audrücke erkennt man, dass gemäß r_1 nach jedem a ein b stehen muss, während gemäß r_2 zwei oder mehr direkt aufeinander folgende a-Zeichen möglich sind.

Beweis:

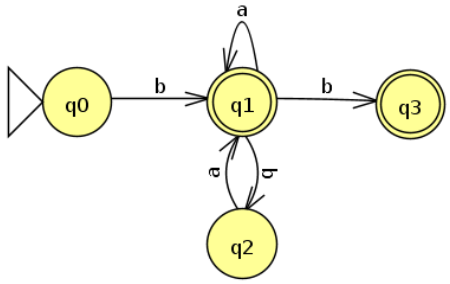
r_1 und r_2 sind äquivalent, g.d.w. $L(r_1) = L(r_2)$.

Da $aab \in L(r_2)$ aber $aab \notin L(r_1)$, sind die Ausrücke nicht äquivalent.

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

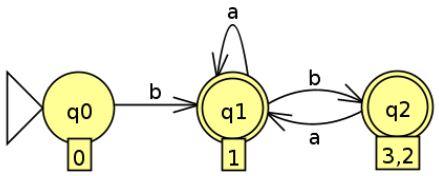
Lösungsansatz:

Zerlege den Ausdruck in drei Teilausdrücke b, $(a \mid ba \mid \varepsilon)^*$ und $(b \mid \varepsilon)$ und konstruiere je einen NEA. Schalte die NEAen in Serie. Wenn man dabei (oder danach) überflüssige Zustände und ε-Übergänge weglässt, erhält man z.B.



Aus dem NEA erhält man durch Teilmengenkonstruktion einen äquivalenten DEA:

DEA-Lösung



d) Geben Sie zu ihrem Automaten aus (c) eine akzeptierende Berechnung an für $babab$

$q0 \overset{b}{\Rightarrow} q1 \overset{a}{\Rightarrow} q1 \overset{b}{\Rightarrow} q2 \overset{a}{\Rightarrow} q1 \overset{b}{\Rightarrow} q2$

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

Punkte ____ von **10**

Gegeben sei folgende kontextfreie Grammatik G

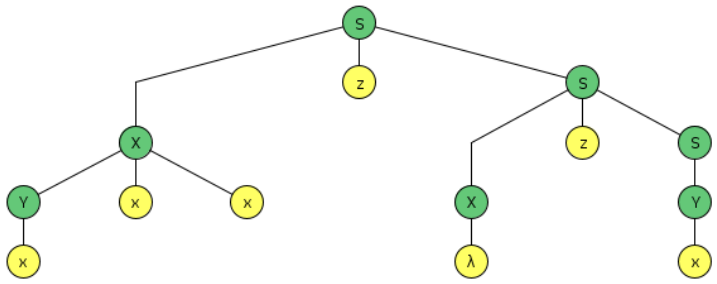
$$S \to XzS \mid Y \mid \varepsilon$$
$$X \to Yxx \mid \varepsilon$$
$$Y \to Yy \mid x$$

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

Lösungsansatz:

Im Eingabewort stehen zwei z-Zeichen. Offensichtlich muss zu deren Erzeugung die Ableitungsregel $S \to XzS$ genau zwei Mal angewendet werden. Der Rest ist dann einfach zu erkennen.

Ableitungsbaum:



Linksableitung:

$\underline{S} \Rightarrow \underline{X}zS \Rightarrow \underline{Y}xxzS \Rightarrow xxxz\underline{S} \Rightarrow xxxz\underline{X}zS \Rightarrow xxxzz\underline{S} \Rightarrow xxxzzx$

b) Geben Sie eine äquivalente Grammatik an, in der das Nonterminal X ersatzlos eliminiert ist.

Lösungsansatz:

X kann nur mit der Regel $S \to XzS$ generiert werden und muss danach durch Yxx oder ε ersetzt werden. Wenn man aus S statt X direkt Yxx oder ε erzeugt, ist X überflüssig geworden und die Regeln für X können entfernt werden:

Statt $S \to XzS, X \to \varepsilon$ erhält man $S \to zS$. Statt $S \to XzS, X \to Yxx$ erhält man $S \to YxxzS$.

Äquivalente Grammatik ohne X:

$$S \to YxxzS \mid zS \mid Y \mid \varepsilon$$
$$Y \to Yy \mid x$$

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

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

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

Aufgabe 4 (4+3+3 Punkte)

Punkte ____ *von* **10**

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.

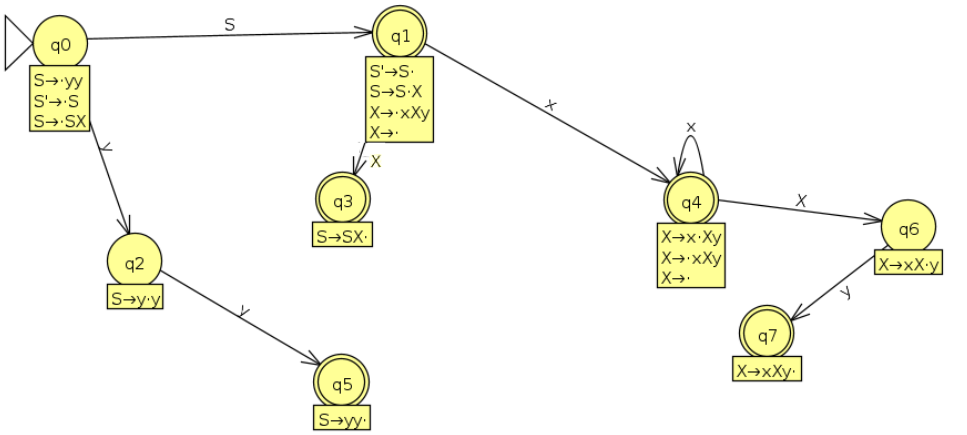
c) Geben Sie die Berechnung des SLR(1)-Parsers für die Eingabe $yyxy$ an. Falls die Tabelle Shift/Reduce-Konflikte enthält, soll der Parser dabei immer die SHIFT-Aktion wählen.

(1) $S \to SX$
(2) $S \to yy$
(3) $X \to xXy$
(4) $X \to \varepsilon$

Mit JFLAP erzeugte Lösung zu (a) und (b):

0	S'	→	S
1	S	→	SX
2	S	→	yy
3	X	→	xXy
4	X	→	λ

	FIRST	FOLLOW
S	{ y }	{ $, x }
X	{ λ, x }	{ $, x, y }



	x	y	$	S	X
0		s2		1	
1	s4 , r4	r4	accept , r4		3
2		s5			
3	r1		r1		
4	s4 , r4	r4	r4		6
5	r2		r2		
6		s7			
7	r3	r3	r3		

(c)

Nr.	Stack	Resteingabe	Aktion
1	0	yyxy	s2
2	0y2	yxy	s5
3	0y2y5	xy	r2
4	0S1	xy	s4
5	0S1x4	y	r4
6	0S1x4X6	y	s7
7	0S1x4X6y7		r3
8	0S1X3		r1
9	0S1		accept

Aufgabe 5 (3 + 7 Punkte)

Punkte ____ von **10**

Betrachten Sie folgende SPL-Definitionen:

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

proc p (ref v1:vektor, i:int) {
    var v2:vektor;
    var j: int;
    j := 2*i;
    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
v2	40	-40	16
j	4	-44	12
FP alt	4	-48	8
RETURN alt	4	-52	4
Argument für printi	4	-56	0

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

Registernutzung der ECO32-VM:

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

```
add     $8,$25,-44          ; $8  <-- Adresse(j)
add     $9,$0,2             ; $9  <-- 2
add     $10,$25,4           ; $10 <-- Adresse(i)
ldw     $10,$10,0           ; $10 <-- Wert(i)
mul     $9,$9,$10           ; $9  <-- 2*Wert(i)
stw     $9,$8,0             ; j   <-- 2*Wert(i)
add     $8,$25,0            ; $8  <-- Adresse(v1)
ldw     $8,$8,0             ; $8  <-- Adresse(Variable des Callers)
add     $9,$25,-44          ; $9  <-- Adresse(j)
ldw     $9,$9,0             ; $9  <-- Wert(j)
add     $10,$0,10           ; $10 <-- Elementanzahl(vektor)
bgeu    $9,$10,_indexError  ; Indexprüfung
mul     $9,$9,4             ; $9  <-- Bytedistanz v1[j] zu v1[0]
add     $8,$8,$9            ; $8  <-- Adresse(v1[j])
ldw     $8,$8,0             ; $8  <-- Wert(v1[j]
stw     $8,$29,0            ; store arg #0
jal     printi
```