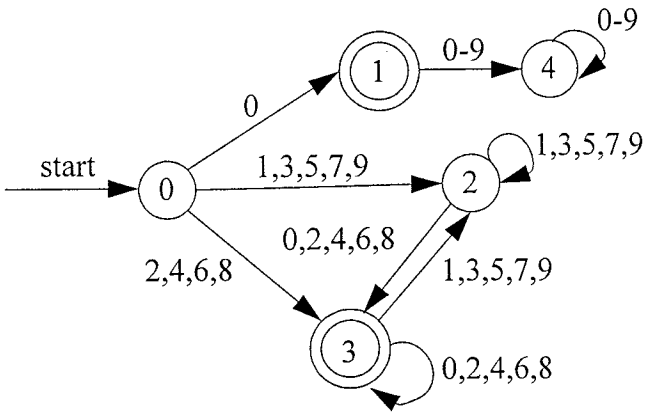
Lösungsvorschläge zur Klausur „1810 Übersetzerbau“

19. Februar 2005

Aufgabe 1

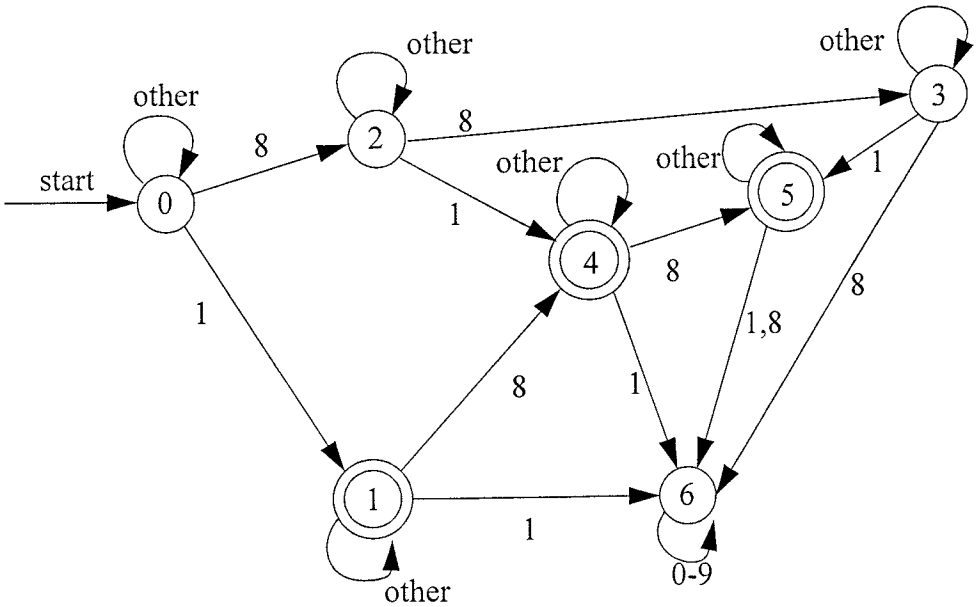
(a)

Der Automat $M_1 = (\{0, 1, 2, 3, 4\}\{0\text{-}9\}, \delta, 0, \{1, 3\})$, wobei δ durch das nachfolgende Diagramm gegeben ist, akzeptiert die Sprache L_1. Er prüft, ob die letzte Ziffer in der Menge {0, 2, 4, 6, 8} vorkommt. Der Sonderfall der führenden Nullen wird durch die Zustände 1 und 4 abgefangen:



(b)

Der Automat $M_2 = (\{0\text{-}6\}\{0\text{-}9\}, \delta, 0, \{1,4,5\})$ zählt die Anzahlen der im Wort enthaltenen 1 und 8. Die Zustände, in denen die geforderten Eigenschaften gelten, sind die Endzustände des Automaten. δ ist durch das folgende Diagram gegeben:



(c)

Die Sprache wird durch den endlichen Automaten

$M = (\{s, x, e, 0, 1, 2, 3, 4, 5, 6\}, \{0\text{-}9\}, \delta, s, \{x, 0\})$

akzeptiert, wobei δ durch die folgende Tabelle beschrieben ist:

Σ / Q	0	1	2	3	4	5	6	7	8	9
s	*x*	1	2	3	4	5	6	0	1	2
x	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*
e	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*	*e*
0	0	1	2	3	4	5	6	0	1	2
1	3	4	5	6	0	1	2	3	4	5
2	6	0	1	2	3	4	5	6	0	1
3	2	3	4	5	6	0	1	2	3	4
4	5	6	0	1	2	3	4	5	6	0
5	1	2	3	4	5	6	0	1	2	3
6	4	5	6	0	1	2	3	4	5	6

Die Zustandsbezeichnung der Zustände $\{0, ..., 6\}$ entsprechen den Werten der bis dahin eingelesenen Zahl modulo 7. Die Zustände x und e dienen der Behandlung von führenden Nullen. Beispielhaft sei der Folgezustand von Zustand 3 bei Eingabe von 2 erläutert. Der Wert der in Zustand 3 bereits gelesenen Zahl kann durch $(7x + 3)$ dargestellt werden. Das Einlesen (Anhängen) der Ziffer 2 ändert diesen Wert zu $(7x + 3) * 10 + 2 = 70x + 32$. Dieser Wert läßt sich durch $7(10x + 4) + 4 = 7y + 4$ darstellen, was den Übergang in Zustand 4 begründet.

Aufgabe 2

(a)

In den Produktionen für die Nichtterminale U und V tritt direkte Linksrekursion auf. Diese können durch folgende Produktionen eliminiert werden:

$$
\begin{aligned}
U &\rightarrow VU' \\
U' &\rightarrow \mathbf{a}VU' \mid \varepsilon \\
W &\rightarrow XW' \\
W' &\rightarrow \mathbf{c}XW' \mid \varepsilon
\end{aligned}
$$

Die Produktionen Y und Z sind nicht erreichbar, müssen also entfallen. Insgesamt erhält man so die folgende Grammatik G':

$G' = (\{U, U', V, W, W'\}, \{\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}\}, P, U)$,

mit $P = \{$
$U \rightarrow VU'$
$U' \rightarrow \mathbf{a}VU' \mid \varepsilon$
$V \rightarrow W \mid \mathbf{b}U\mathbf{b}$
$W \rightarrow XW'$
$W' \rightarrow \mathbf{c}XW' \mid \varepsilon$
$X \rightarrow \mathbf{d} \mid \mathbf{e}W\mathbf{e}$ $\}$.

(b)

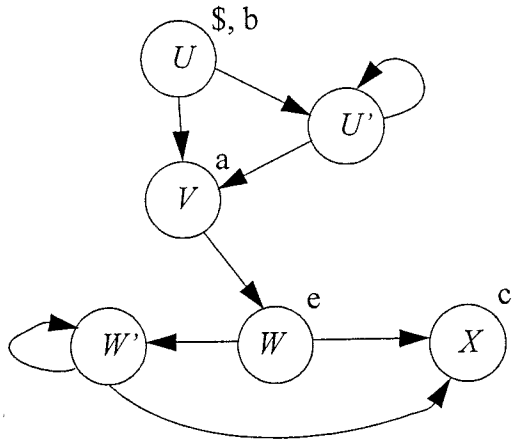
Die Berechnung der FIRST-Mengen und der initialen Steuermengen ergibt:

{**b**, **d**, **e**}	U	$\rightarrow$	VU'	{**b**, **d**, **e**}	(1)
{**a**, ε}	U'	$\rightarrow$	$\mathbf{a}VU' \mid$	{**a**}	(2)
			ε	{ε}	(3)
{**b**, **d**, **e**}	V	$\rightarrow$	$W \mid$	{**d**, **e**}	(4)
			$\mathbf{b}U\mathbf{b}$	{**b**}	(5)
{**d**, **e**}	W	$\rightarrow$	XW'	{**d**, **e**}	(6)
{**c**, ε}	W'	$\rightarrow$	$\mathbf{c}XW' \mid$	{**c**}	(7)
			ε	{ε}	(8)
{**d**, **e**}	X	$\rightarrow$	$\mathbf{d} \mid$	{**d**}	(9)
			$\mathbf{e}W\mathbf{e}$	{**e**}	(10)

Zur Berechnung der FOLLOW-Mengen ist es wichtig Produktionen der Form $A \rightarrow \alpha B\beta$, mit $\beta \neq \varepsilon$ und $A \rightarrow \alpha B$ zu identifizieren ($\alpha = \varepsilon$ ist zulässig):

Produktion	Form(en)	α, B, β	Markierung	Kante(n)
$U \rightarrow VU'$	$A \rightarrow \alpha B\beta$ $A \rightarrow \alpha B$	ε, V, U' V, U'	V {**a**}	$U \rightarrow V$ $U \rightarrow U'$
$U' \rightarrow \mathbf{a}VU'$	$A \rightarrow \alpha B\beta$ $A \rightarrow \alpha B$	**a**, V, U' **a**V, U'	V {**a**}	$U' \rightarrow V$ $U' \rightarrow U'$
$V \rightarrow W$	$A \rightarrow \alpha B$	ε, W		$V \rightarrow W$
$V \rightarrow \mathbf{b}U\mathbf{b}$	$A \rightarrow \alpha B\beta$	**b**, U, **b**	U {**b**}	
$W \rightarrow XW'$	$A \rightarrow \alpha B\beta$ $A \rightarrow \alpha B$	ε, X, W' X, W'	X {**c**}	$W \rightarrow X$ $W \rightarrow W'$
$W' \rightarrow \mathbf{c}XW'$	$A \rightarrow \alpha B\beta$ $A \rightarrow \alpha B$	**c**, X, W' X, W'	X {**c**}	$W' \rightarrow X$ $W' \rightarrow W'$
$X \rightarrow \mathbf{e}W\mathbf{e}$	$A \rightarrow \alpha B\beta$	**e**, W, **e**	W {**e**}	

Damit erhält man den folgenden Graphen:



Propagiert man nun die Kantenmarkierungen, so erhält man die folgenden FOLLOW-Mengen für *U'* und *W'*:

FOLLOW(*U'*) = {**$, b**}

FOLLOW(*W'*) = {**$, a, b, e**}

Konkateniert man nun die FIRST und FOLLOW-Mengen, so erhält man folgende Steuermengen für *G'*:

D(1)={**b, d, e**}	*D*(2)={**a**}	*D*(3)={**$, b**}
D(4)={**d, e**}	*D*(5)={**b**}	*D*(6)={**d, e**}
D(7)={**c**}	*D*(8)={**$, a, b, e**}	*D*(9)={**d**}
D(10)={**e**}		

Damit sind alle Steuermengen disjunkt und *G'* ist eine Grammatik des Typs LL(1).

Aufgabe 3

(a)

Die Tabelle sieht folgendermaßen aus:

		Eingabe							
		⊗	©	®	⊕	(	)	**id**	**$**
Stack	⊗	·>	·>	·>	·>	<·	·>	<·	·>
	©	<·	·>	·>	·>	<·	·>	<·	·>
	®	<·	<·	·>	·>	<·	·>	<·	·>
	⊕	<·	<·	<·	<·	<·	·>	<·	·>
	(	<·	<·	<·	<·	<·	≐	<·	
	)	·>	·>	·>	·>		·>		·>
	id	·>	·>	·>	·>		·>		·>
	$	<·	<·	<·	<·	<·		<·	

(b)

Stack		*Eingabe*	*Aktion*
$	<·	**(id⊕id⊕id)©id⊗id$**	*shift*
$<·(	<·	**id⊕id⊕id)©id⊗id$**	*shift*
$<·(<·**id**	·>	**⊕id⊕id)©id⊗id$**	*reduce* mit $E \rightarrow$ **id**
$<·(*E*	<·	**⊕id⊕id)©id⊗id$**	*shift*
$<·(*E*<·⊕	<·	**id⊕id)©id⊗id$**	*shift*
$<·(*E*<·⊕<·**id**	·>	**⊕id)©id⊗id$**	*reduce* mit $E \rightarrow$ **id**
$<·(*E*<·⊕*E*	<·	**⊕id)©id⊗id$**	*shift*
$<·(*E*<·⊕*E*<·⊕	<·	**id)©id⊗id$**	*shift*
$<·(*E*<·⊕*E*<·⊕<·**id**	·>	**)©id⊗id$**	*reduce* mit $E \rightarrow$ **id**
$<·(*E*<·⊕*E*<·⊕*E*	·>	**)©id⊗id$**	*reduce* mit $E \rightarrow E \oplus E$
$<·(*E*<·⊕*E*	·>	**)©id⊗id$**	*reduce* mit $E \rightarrow E \oplus E$
$<·(*E*	≐	**)©id⊗id$**	*shift*

$<·(E≐)	·>	©id⊗id$	*reduce* mit $E \rightarrow (\ E\ )$
$E	<·	©id⊗id$	*shift*
$E<·©	<·	id⊗id$	*shift*
$E<·©<·id	·>	⊗id$	*reduce* mit $E \rightarrow$ **id**
$E<·©E	<·	⊗id$	*shift*
$E<·©E<·⊗	<·	id$	*shift*
$E<·©E<·⊗<·id	·>	$	*reduce* mit $E \rightarrow$ **id**
$E<·©E<·⊗E	·>	$	*reduce* mit $E \rightarrow E \otimes E$
$E<·©E	·>	$	*reduce* mit $E \rightarrow E \copyright E$
$E	·>	$	*accept*

Aufgabe 4

(a)

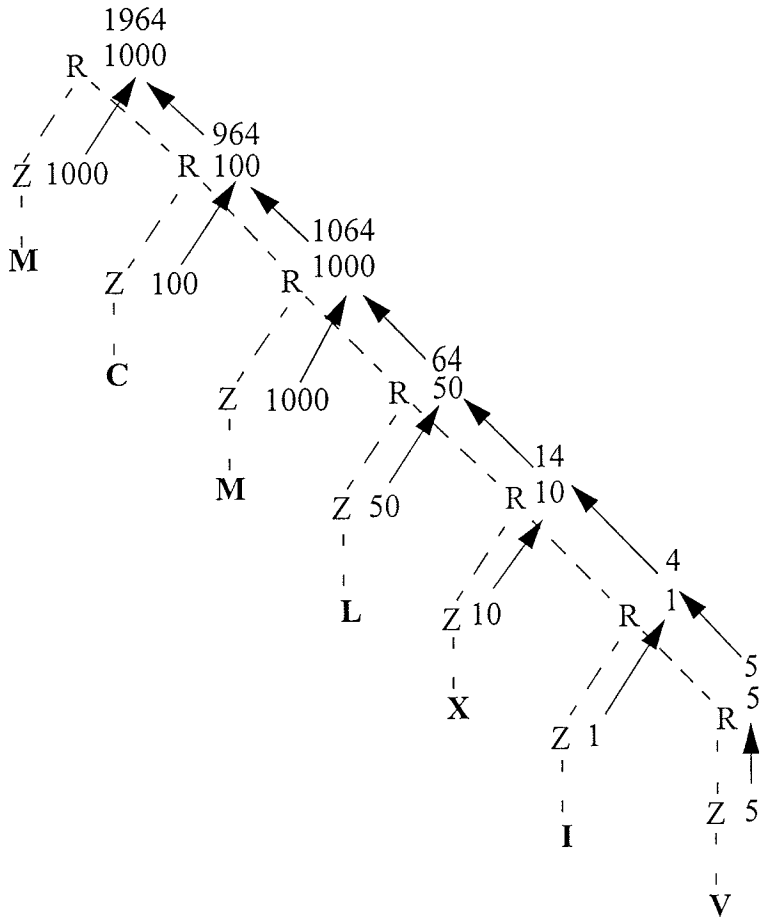
Für einzelne Ziffern merken wir uns den Wert im Attribut *val*. Für Sequenzen von Ziffern wird zusätzlich der Wert der letzten Ziffer im Attribut *last* gespeichert.

Produktion	*Semantische Regel*

$R \rightarrow ZR_1$	**if**($Z.val >= R_1.last$){
	$R.val := Z.val + R_1.val$
	}**else**{
	$R.val := R_1.val - Z.val$
	}
	$R.last := Z.val$

$R \rightarrow Z$	$R.val := Z.val$ $R.last := Z.val$
$Z \rightarrow$ **I**	$Z.val := 1$
$Z \rightarrow$ **X**	$Z.val := 10$
$Z \rightarrow$ **C**	$Z.val := 100$
$Z \rightarrow$ **M**	$Z.val := 1000$
$Z \rightarrow$ **V**	$Z.val := 5$
$Z \rightarrow$ **L**	$Z.val := 50$
$Z \rightarrow$ **D**	$Z.val := 500$

(b)



Aufgabe 5

(a)

Der Code für die Prozedur lautet:

```
t := b * b
x := a + t
return y
```

Der Code der Prozedur main lautet:

```
a := 5
b := 7
valparam a
valparam b
call p
getresult y
x := y
```

(b)

Prozedurtabelle:

name	*s_depth*	*static_size*	*start*	*typeind*
compute	1	32	1456	1

- *name*: Name der Prozedur.
- *s_depth*: Statische Schachtelungstiefe (*static depth*) der Prozedur im Programmtext.
- *static_size*: Größe eines Prozedurrahmens, dieser Berechnet sich aus einem festen Header (im Kurstext 16 Bytes) sowie den Parametern (8 Bytes) und der Größe der lokalen Variablen (8 Bytes). Bei unterschiedlichen Datentypen kann sich bedingt durch Alignments sogar noch ein das reine Datenvolumen übersteigender Wert ergeben.
- *start*: Adresse im Codespeicher.
- *typeind*: Typ des Rückgabewertes dargestellt als Indexwert der Typtabelle.

Symboltabelle:

type	*name*	*s_depth*	*offset*	*size*	*value*	*align*	*typeind*
const	a	0	16	4	5	4	1
const	b	0	20	4	7	4	1
var	x	0	24	4	-	4	1
var	a	1	16	4	-	4	1
var	b	1	20	4	-	4	1
var	t	1	24	4	-	4	1
var	x	1	28	4	-	4	1

- *type*: Diese Feld unterscheidet zwischen Konstanten und Variablen.
- *offset*: Anfangsposition relativ zur Anfangsadresse des Rahmens.
- *size*: Größe des Datentyps.
- *value*: Momentan zugewiesener Wert.
- *alignment*: Kann nur auf einer durch diesen Wert teilbaren Adresse beginnen.

Bemerkung: *size* und *alignment* könnten natürlich auch in der Typtabelle verwaltet werden, da diese nur vom Datentyp und nicht von einem speziellen Symbol abhängen.

Aufgabe 6 (Maschinen-Code-Optimierung)

(a)

Im Programm wird x als Zählvariable für die Schleife verwendet. Sämtliche Zeilen, die mit der Inkrementierung des Wertes zu tun haben (9-11), können entfernt werden. Stattdessen wird eine Zeile `INC x` eingefügt. Am Speicherort t wird der Wert 2 gehalten, der stets zur Multiplikation verwendet wird. Diese Multiplikation kann nun direkt mit dem SHLFT-Befehl durchgeführt werden. Dadurch verschwinden die Zeilen 3, 4, 14 und 15. Neu eingefügt wird dafür der Befehl `SHLFT T`.

Das Ergebnisprogramm lautet (nach Umnumerierung):

```
 1      LOAD  S,10
 2      STORE S,y
 3      LOAD  T,0
 4      STORE T,z
 5      STORE T,x
 6  L1: if x > y goto L2
 7      INC x
 8      LOAD  T,z
 9      SHLFT T
10      STORE T,z
11      goto L3
12  L2: LOAD  T,z
13      LOAD  U,z
14      MULT  T,U
15      goto L4
16  L3: goto L1
17  L4: STORE T,z
```

(b)

Durch Peephole-Optimierung erkennen wir nun, daß bereits zu Beginn der Wert von z in T liegt und in jedem Schleifendurchlauf geladen, verändert und wieder gespeichert wird. Das mehrfache Laden und Speichern ist unnötig. Stattdessen wird z die ganze Zeit in T gehalten und erst am Ende gespeichert. Dadurch entfallen die Zeilen 4, 8, 10 und 12. Zeile 13 wird, da der Registerinhalt von T ja nun nicht mehr in z gespeichert wird, ebenfalls gelöscht und Zeile 14 in `MULT T,T` geändert.

Abschließend kann man noch die überflüssigen Sprünge entfernen. Die Sprungmarke L3 wird entfernt und der Sprungbefehl in 11 durch `goto L1` ersetzt. Die Sprungmarke L4 wird damit genauso überflüssig wie der Sprungbefehl in 15. Beide werden entfernt.

Das endgültige Programm hat dann folgende Form:

```
 1      LOAD  S,10
 2      STORE S,y
 3      LOAD  T,0
 4      STORE T,x
 5  L1: if x > y goto L2
 6      INC x
```

```
 7       SHLFT T
 8       goto L1
 9  L2:  MULT  T,T
10       STORE T,z
```

(c)

In z steht am Ende eine 0. In der Schleife wird der initiale Wert (ebenfalls 0) 10mal mit 2 multipliziert und am Ende noch einmal mit sich selbst. Letztlich bleibt der Wert aber 0.