Lösungsvorschläge zur Klausur „1810 Übersetzerbau“

17. Februar 2007

Aufgabe 1

Eine Spezifikation für Lex könnte etwa folgendermaßen aussehen:

```
 /*** C-DEKLARATIONEN ***/
%{
#include <iostream>
using namespace std;
%}

 /*** MUSTER-DEFINTIONEN ***/

letter    [a-zA-Z]
digit     [0-9]
alnum     {letter}|{digit}

 /*
  *  Hilfsdefinitionen für Leerraum bzw. alle nichtnumerischen
  *  Zeichen (nan: not a number).
  */

empty       [ \t\n]
nan         [^a-zA-Z]|\n

triple      {digit}{digit}?|[0-2][0-4][0-9]|[0-2][5][0-5]
alnum2      {alnum}|"-"
alnum3      {alnum}{alnum2}*

ipNumber    {triple}\.{triple}\.{triple}\.{triple}

 /*
  *  Eine Domain enthält mindestens 2 Punkte, in E-Mail Addressen
  *  auch weniger.
  */

top         [a-zA-Z]{alnum2}*
domain      {alnum3}\.{alnum3}(\.{alnum3})*\.{top}
domain2     {alnum3}(\.{alnum3})*

email {alnum3}(\.|{alnum2})*{empty}*("@"|"(at)"){empty}*{domain2}

bank    [Bb]([Ll][Zz]|[Aa][Nn][Kk])
konto   [Kk]([Oo][Nn])?[Tt][Oo]

bankaccount    {bank}{nan}+{digit}+{nan}+{konto}{nan}+{digit}+
```

```
%% /*** AKTIONEN ***/

{ipNumber}|{domain}      { cout << "Host: " << yytext << endl; }

{bankaccount}            { cout << "Bank account: "
                               << yytext << endl;  }

{email}                  { cout << "E-Mail: " << yytext << endl; }

.|\n                     { /* Ausgabe nicht auf ein Muster
                            * passender Zeichen verhindern.
                           */                                  }

%%  /*** HAUPTPROGRAMM ***/
int main() {

  return yylex();

}
```

Aufgabe 2

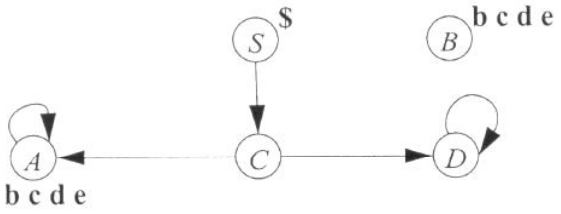
(a)

Die initialen Steuermengen sind:

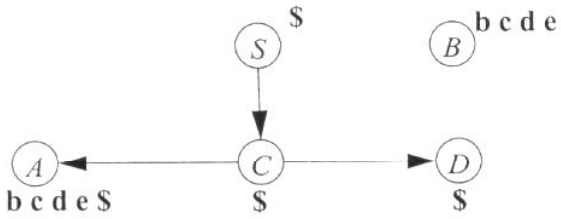
$S \rightarrow$	ABC	{**a**, **b**, **c**, **d**, **e**}
$A \rightarrow$	**a**A	{**a**}
$A \rightarrow$	ε	{ε}
$B \rightarrow$	**b**	{**b**}
$B \rightarrow$	ε	{ε}
$C \rightarrow$	D	{**b**, **d**, **e**}
$C \rightarrow$	**c**A	{**c**}
$D \rightarrow$	**d**D	{**d**}
$D \rightarrow$	B**e**	{**b**, **e**}

(b)

Der initiale Graph (ohne Propagation) der Knotenmarkierungen ist:



nach Propagation der Markierungen ergibt sich folgendes Bild:



(c)

Damit ergeben sich die folgenden Steuermengen:

S	$\rightarrow$	ABC	{**a**, **b**, **c**, **d**, **e**}
A	$\rightarrow$	**a**A	{**a**}
A	$\rightarrow$	ε	{**b**, **c**, **d**, **e**, **$**}
B	$\rightarrow$	**b**	{**b**}
B	$\rightarrow$	ε	{**b**, **c**, **d**, **e**}
C	$\rightarrow$	D	{**b**, **d**, **e**}
C	$\rightarrow$	**c**A	{**c**}
D	$\rightarrow$	**d**D	{**d**}
D	$\rightarrow$	B**e**	{**b**, **e**}

Die Grammatik G ist nicht LL(1), da die Steuermengen der Produktionen von B nicht disjunkt sind : $\{\mathbf{b}\} \cap \{\mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}\} = \{\mathbf{b}\} \neq \emptyset$.

Aufgabe 3

Die Vorrang-Tabelle sieht folgendermaßen aus:

Stack \ *Eingabe*	\	∩	∪	×	(	)	{	}	**elem**	**id**	$
\	<·	·>	·>	·>	<·	·>	<·			<·	·>
∩	<·	·>	·>	·>	<·	·>	<·			<·	·>
∪	<·	<·	·>	·>	<·	·>	<·			<·	·>
×	<·	<·	<·	·>	<·	·>	<·			<·	·>
(	<·	<·	<·	<·	<·	≐	<·			<·	
)	·>	·>	·>	·>		·>					·>
{								≐	<·		
}							·>				·>
elem								·>			
id	·>	·>	·>	·>		·>			<·	<·	·>
$	<·	<·	<·	<·	<·		<·			<·	

Die Eingabe wird wie folgt vom Parser verarbeitet:

Stack		*Eingabe*	*Aktion*
$	<·	**(id \ id \ id ∪ id) × { elem elem }**$	*shift*
$<·(	<·	**id \ id \ id ∪ id) × { elem elem }**$	*shift*
$<·(<·**id**	·>	**\ id \ id ∪ id) × { elem elem }**$	*reduce* mit $E \to$ **id**
$<·(*E*	<·	**\ id \ id ∪ id) × { elem elem }**$	*shift*
$<·(*E*<· \	<·	**id \ id ∪ id) × { elem elem }**$	*shift*
$<·(*E*<· \ <· **id**	·>	**\ id ∪ id) × { elem elem }**$	*reduce* mit $E \to$ **id**
$<·(*E*<· \ *E*	<·	**\ id ∪ id) × { elem elem }**$	*shift*
$<·(*E*<· \ *E* <· \	<·	**id ∪ id) × { elem elem }**$	*shift*
$<·(*E*<· \ *E* <· \ <·**id**	·>	**∪ id) × { elem elem }**$	*reduce* mit $E \to$ **id**
$<·(*E*<· \ *E* <· \ *E*	·>	**∪ id) × { elem elem }**$	*reduce* mit $E \to E \setminus E$

$\$ \lessdot \cdot (E \lessdot \cdot \setminus E$	$\cdot>$	$\cup$ **id**) $\times$ { **elem elem** }$	*reduce* mit $E \to E \setminus E$
$\$ \lessdot \cdot (E$	$\lessdot \cdot$	$\cup$ **id**) $\times$ { **elem elem** }$	*shift*
$\$ \lessdot \cdot (E \lessdot \cdot \cup$	$\lessdot \cdot$	**id**) $\times$ { **elem elem** }$	*shift*
$\$ \lessdot \cdot (E \lessdot \cdot \cup \lessdot \cdot$ **id**	$\cdot>$	) $\times$ { **elem elem** }$	*reduce* mit $E \to$ **id**
$\$ \lessdot \cdot (E \lessdot \cdot \cup E$	$\cdot>$	) $\times$ { **elem elem** }$	*reduce* mit $E \to E \cup E$
$\$ \lessdot \cdot (E$	$\doteq$	) $\times$ { **elem elem** }$	*shift*
$\$ \lessdot \cdot (E \doteq)$	$\cdot>$	$\times$ { **elem elem** }$	*reduce* mit $E \to (E)$
$\$ E$	$\lessdot \cdot$	$\times$ { **elem elem** }$	*shift*
$\$ E \lessdot \cdot \times$	$\lessdot \cdot$	{ **elem elem** }$	*shift*
$\$ E \lessdot \cdot \times \lessdot \cdot \{$	$\lessdot \cdot$	**elem elem** }$	*shift*
$\$ E \lessdot \cdot \times \lessdot \cdot \{ \lessdot \cdot$ **elem**	$\cdot>$	**elem** }$	*reduce* mit $F \to$ **elem**
$\$ E \lessdot \cdot \times \lessdot \cdot \{ \lessdot \cdot F$	$\lessdot \cdot$	**elem** }$	*shift*
$\$ E \lessdot \cdot \times \lessdot \cdot \{ \lessdot \cdot F \lessdot \cdot$ **elem**	$\cdot>$	}$	*reduce* mit $F \to F$ **elem**
$\$ E \lessdot \cdot \times \lessdot \cdot \{ F$	$\doteq$	}$	*shift*
$\$ E \lessdot \cdot \times \lessdot \cdot \{ F \doteq \}$	$\cdot>$	$	*reduce* mit $E \to \{ F \}$
$\$ E \lessdot \cdot \times E$	$\cdot>$	$	*reduce* mit $E \to E \times E$
$\$ E$	$\cdot>$	$	*accept*

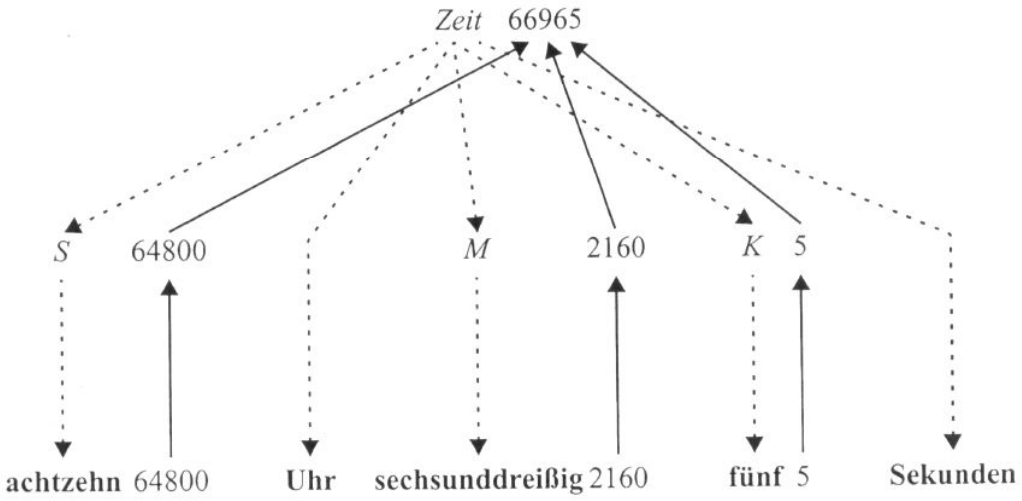
Aufgabe 4

(a)

Das synthetisierte Attribut *sek*, das die Anzahl der vergangenen Sekunden pro Tag enthält, wird zusätzlich zu den Grammatiksymbolen eingeführt. Die Berechnung erfolgt dann wie nachfolgend gezeigt:

Produktion			semantische Regel	
Zeit	→	*S* **Uhr** *M* **und** *K* **Sekunden**	*Zeit.sek*	:= *S.sek* + *M.sek* + *K.sek*
S	→	**null** | **ein** | ... | **dreiundzwanzig**	*S.sek*	:= 0 | 1 · 3600 | ... | 23 · 3600
M	→	ε	*M.sek*	:= 0
M	→	**eins** | **zwei** | ... | **neunundfünfzig**	*M.sek*	:= 1 · 60 | 2 · 60 | ... | 59 · 60
K	→	ε	*K.sek*	:= 0
K	→	**eine** | **zwei** | ... | **neunundfünfzig**	*K.sek*	:= 1 | 2 | ... | 59

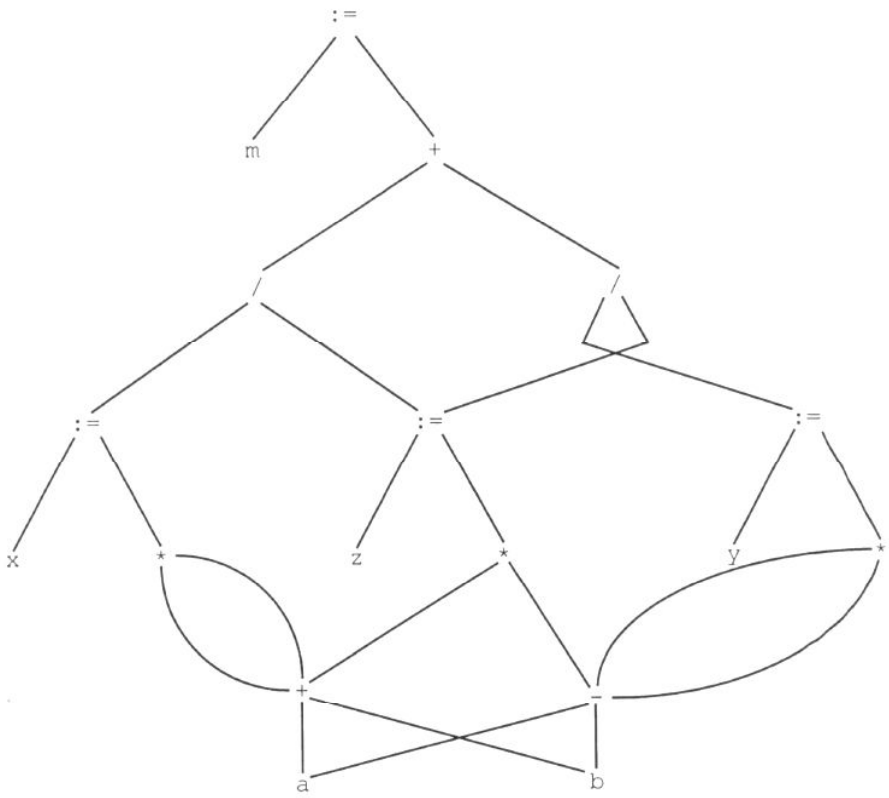
(b)



Aufgabe 5

(a)

Der DAG hat folgende Form:



(b)

Die Postfix-Notation für den gegebenen Code sieht wie folgt aus:

```
(1)  x a b + a b + * :=
(2)  y a b - a b - * :=
(3)  z a b + a b - * :=
(4)  m x z / y z / + :=
```

Dabei ist der Postfix-Code nicht mehr zwangsläufig in Zeilen zu schreiben, sondern läßt sich direkt hintereinander ausführen, wie man bei der Auswertung in der Stackmaschine sieht.

Code	*Stack*
`lvar x`	`[x]`
`var a`	`[x] a`
`var b`	`[x] a b`
`+`	`[x] a+b`
`var a`	`[x] a+b a`

Code	*Stack*	
`var b`	`[x] a+b a b`	
`+`	`[x] a+b a+b`	
`*`	`[x] (a+b)*(a+b)`	
`:=`		Speichern des berechneten Wertes an der Adresse `x`
`lvar y`	`[y]`	
`var a`	`[y] a`	
`var b`	`[y] a b`	
`-`	`[y] a-b`	
`var a`	`[y] a-b a`	
`var b`	`[y] a-b a b`	
`-`	`[y] a-b a-b`	
`*`	`[y] (a-b)*(a-b)`	
`:=`		Speichern des berechneten Wertes an der Adresse `y`
`lvar z`	`[z]`	
`var a`	`[z] a`	
`var b`	`[z] a b`	
`+`	`[z] a+b`	
`var a`	`[z] a+b a`	
`var b`	`[z] a+b a b`	
`-`	`[z] a+b a-b`	
`*`	`[z] (a+b)*(a-b)`	
`:=`		Speichern des berechneten Wertes an der Adresse `z`
`lvar m`	`[m]`	
`var x`	`[m] x`	
`var z`	`[m] x z`	
`/`	`[m] x/z`	
`var y`	`[m] x/z y`	
`var z`	`[m] x/z y z`	
`/`	`[m] x/z y/z`	
`+`	`[m] (x/z)+(y/z)`	
`:=`		Speichern des berechneten Wertes an der Adresse `m`

(c)

Bei der Übersetzung in 3AC werden hier gleiche Teilausdrücke nicht mehrfach übersetzt.

```
(1)  s := a + b
(2)  t := a - b
(3)  x :- s * s
(4)  y := t * t
(5)  z := s * t
(6)  s := x / z
(7)  t := y / z
(8)  z := s + t
```

Aufgabe 6

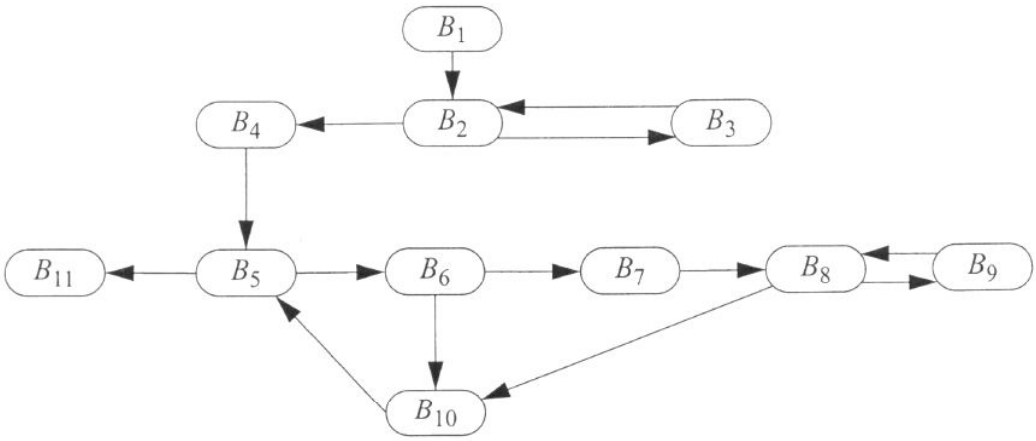
(a)

Die Basisblöcke umfassen die nachfolgend genannten Zeilen:

Block	**Zeilen**	**Begründung für den Blockanfang**
B_1	1-2	Programmanfang
B_2	3	Ziel von `goto` in Zeile 8
B_3	4-8	nach Verzweigung
B_4	9-12	Ziel von `goto` in Zeile 3
B_5	13	Ziel von `goto` in Zeile 28
B_6	14-17	nach Verzweigung
B_7	18-20	nach Verzweigung
B_8	21	Ziel von `goto` in Zeile 26
B_9	22-26	nach Verzweigung
B_{10}	27-28	Ziel von `goto` in Zeilen 17 und 21
B_{11}	29	Ziel von `goto` in Zeile 13

(b)

Es ergibt sich der folgende Flußgraph:



(c)

Das Programm enthält die folgenden Definitionen für die relevanten Variablen:

Variable	**Zeile**	**Block**
`i`	2	B_1
	7	B_3
`j`	20	B_7
	25	B_9
`k`	12	B_4
	27	B_{10}
`max`	1	B_1

Wir erhalten die folgenden Mengen:

Block	**in**	**gen**	**kill**	**out**
B_1	$\varnothing$	{1, 2}	{7}	{1, 2}
B_2	{1, 2, 7}	$\varnothing$	$\varnothing$	{1, 2, 7}
B_3	{1, 2, 7}	{7}	{2}	{1, 7}
B_4	{1, 2, 7}	{12}	{27}	{1, 2, 7, 12}
B_5	{1, 2, 7, 12, 20, 25, 27}	$\varnothing$	$\varnothing$	{1, 2, 7, 12, 20, 25, 27}

B_6	{1, 2, 7, 12, 20, 25, 27}	∅	∅	{1, 2, 7, 12, 20, 25, 27}
B_7	{1, 2, 7, 12, 20, 25, 27}	{20}	{25}	{1, 2, 7, 12, 20, 27}
B_8	{1, 2, 7, 12, 20, 25, 27}	∅	∅	{1, 2, 7, 12, 20, 25, 27}
B_9	{1, 2, 7, 12, 20, 25, 27}	{25}	{20}	{1, 2, 7, 12, 25, 27}
B_{10}	{1, 2, 7, 12, 20, 25, 27}	{27}	{12}	{1, 2, 7, 20, 25, 27}
B_{11}	{1, 2, 7, 12, 20, 25, 27}	∅	∅	{1, 2, 7, 12, 20, 25, 27}