Lösungsvorschläge zur 1. Übung „Compilerbau“, SS 2004

Aufgabe 1

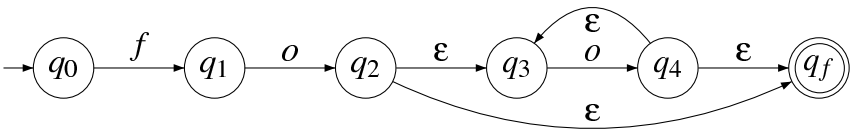
a) $\alpha_{\pm?} := \{+,-\} \vee \Lambda^\star$
$\alpha_d := \{0,...,9\}$
$\alpha_d^+ := \alpha_d \alpha_d^\star$
$\alpha_s := \{e,s,f,d,l\}\alpha_{\pm?}\alpha_d^+$

$\alpha := \alpha_{\pm?}(\alpha_d^+\alpha_s \vee (\{.\}\alpha_d^+ \vee \alpha_d^+\{.\}\alpha_d^\star)(\alpha_s \vee \Lambda^\star))$

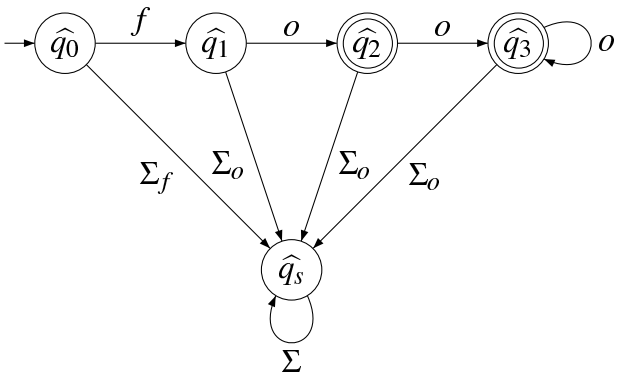
b) $w_1 = 1e9$, $w_2 = .01$, $w_3 = 129.1f + 10$, $v_1 = 1$, $v_2 = .e2$, $v_3 = 129. + 1$

Aufgabe 2

a) $\mathfrak{A}(\beta_0) = \mathfrak{A}(foo^\star)$:

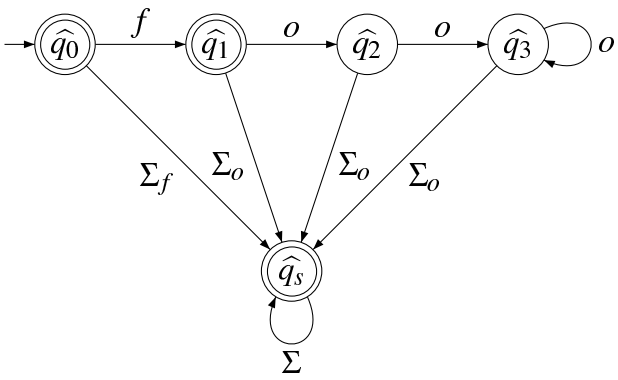


b) Zuerst wird der NFA in einen DFA transformiert (z.B. mittels Potenzmengenkonstruktion). Am Beispiel aus der Aufgabenstellung mit $\Sigma_a := \Sigma \backslash \{a\}$ für ein $a \in \Sigma$ ergibt sich:

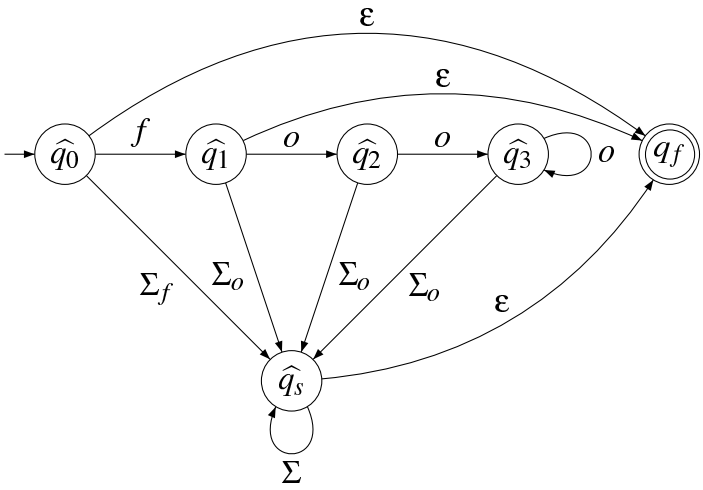


wobei $\widehat{q_0} := \{q_0\}$, $\widehat{q_1} := \{q_1\}$, $\widehat{q_2} := \{q_2,q_3,q_f\}$, $\widehat{q_3} := \{q_4,q_3,q_f\}$, $\widehat{q_s} := \emptyset$.

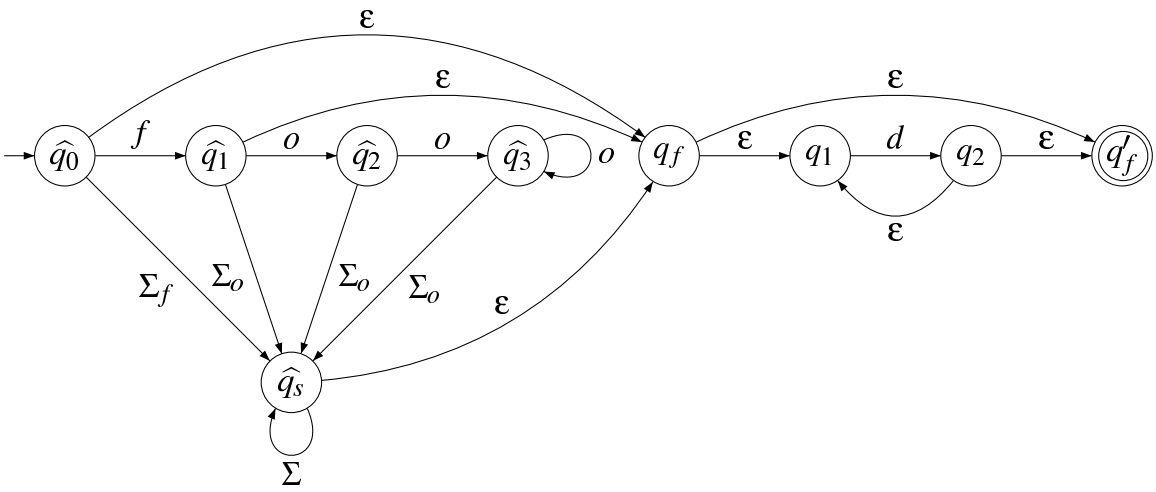
Um den Komplementautomaten zu erhalten, wird die Endzustandsmenge eines DFA invertiert (d.h. $F_{neu} = Q \backslash F_{alt}$). Man erhält also $\mathfrak{A}(\neg\beta_0)$:



Um einen „Thompson-kompatiblen“ Automaten zu erhalten führt man einen neuen Zustand als einzigen Endzustand ein, den man über ε-Transitionen von allen vorherigen Endzuständen erreichen kann. Es ergibt sich:



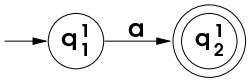
c) $\mathfrak{A}(\beta)$:



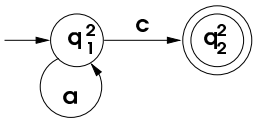
d) Der Automat erkennt das Wort *w* mit dem folgenden akzeptierenden Lauf:
$\widehat{q_0} \xrightarrow{f} \widehat{q_1} \xrightarrow{o} \widehat{q_2} \xrightarrow{o} \widehat{q_3} \xrightarrow{l} \widehat{q_s} \xrightarrow{e} \widehat{q_s} \xrightarrow{\varepsilon} q_f \xrightarrow{\varepsilon} q_1 \xrightarrow{d} q_2 \xrightarrow{\varepsilon} q'_f$

Aufgabe 3

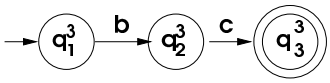
a) $\mathfrak{A}_1$:



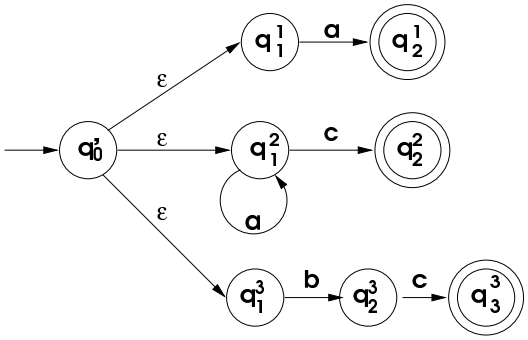
$\mathfrak{A}_2$:



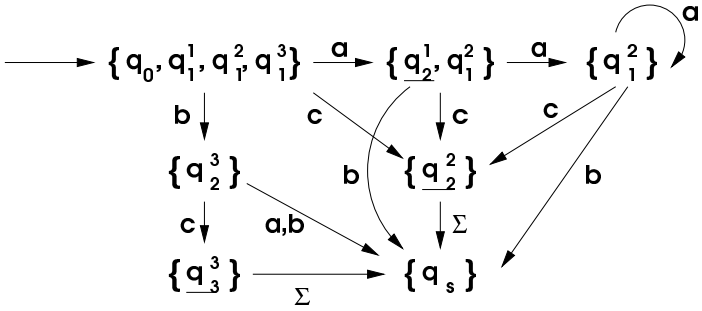
$\mathfrak{A}_3$:



Konstruktion des Vereinigungsautomaten (NFA). Neuer Anfangszustand q_0', Menge der Zustände $Q = \dot{\bigcup}_{i=0}^{3} Q_i$ mit $Q_0 = \{q_0'\}$, Menge der Endzustände $F = \dot{\bigcup}_{i=1}^{3} F_i$.



Dann Konstruktion des DFA mittels Potenzmengenkonstruktion:



Benenne Zustände um:

$$q_0 := \{q_0', q_1^1, q_1^2, q_1^3\} \qquad q_1 := \{q_2^1, q_1^2\}$$
$$q_2 := \{q_1^2\} \qquad q_3 := \{q_2^2\}$$
$$q_4 := \{q_2^3\} \qquad q_5 := \{q_3^3\}$$

$$F^{(1)} = \{q_1\}, \qquad F^{(2)} = \{q_3\}, \qquad F^{(3)} = \{q_5\}$$

b) $P = Q \setminus \{q_s\}$

c)

$$
\begin{aligned}
&(N, q_0aaabc, \varepsilon) \vdash \\
&(T_1, q_1aabc, \varepsilon) \vdash \\
&(T_1, aq_2abc, \varepsilon) \vdash \\
&(T_1, aaq_2bc, \varepsilon) \vdash \\
&(N, q_0aabc, T_1) \vdash \\
&(T_1, q_1abc, T_1) \vdash \\
&(T_1, aq_2bc, T_1) \vdash \\
&(N, q_0abc, T_1T_1) \vdash \\
&(T_1, q_1bc, T_1T_1) \vdash \\
&(N, q_0bc, T_1T_1T_1) \vdash \\
&(N, q_4c, T_1T_1T_1) \vdash \\
&(T_3, q_5, T_1T_1T_1)
\end{aligned}
$$

Ausgabe: $T_1T_1T_1T_3$

d) $(n+1)+(n-1+1)+(n-2+1)+\ldots+1 \rightsquigarrow O(n^2)$

Lösungsvorschläge zur 2. Übung „Compilerbau“, SS 2004

Aufgabe 4

a) Gegenbeispiel:

$$w = aaaa \qquad \alpha_1 = aa \qquad \alpha_2 = aaa$$

Offensichtlich ist $(w_1, w_2) = (aa, aa)$ eine Zerlegung von w, aber *keine* lm-Zerlegung, weil $w_1 \in [\![\alpha_1]\!]$ und es existiert eine Verlängerung $w_1 a \in [\![\alpha_2]\!]$.

b) kein großer Nutzen, da in heutigen Programmiersprachen Symbole auf kurzen lookahead ausgelegt sind (*whitespace* als Trenner; z.B. reicht in Pascal *lookahead* von 2), , daher wenig backtracking.

Aufgabe 5

Gegeben seien die Automaten

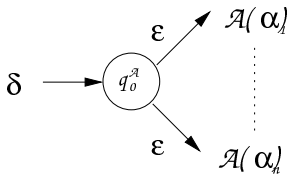
$$\mathfrak{A}(\alpha_1), \ldots, \mathfrak{A}(\alpha_n), \qquad \mathfrak{A}(\alpha_i) = \langle Q_i, \Sigma, \delta_i, q_0^i, F_i \rangle \in \mathrm{NFA}(\Sigma)$$

a) Konstruiere

$$\mathfrak{A} = \left\langle Q, \Sigma, \delta, q_0^{\mathfrak{A}}, F^{\mathfrak{A}} \right\rangle \in \mathrm{NFA}(\Sigma)$$

mit

$$Q_0 = \{q_0^{\mathfrak{A}}\} \qquad Q = \dot{\bigcup_{i=0}^{n}} Q_i \qquad F^{\mathfrak{A}} = \dot{\bigcup_{i=1}^{n}} F_i$$



Konstruktion des Backtrack-Automaten nach NFA-Methode:

$$F^{(i)} \subseteq \mathcal{P}(Q), \quad F = \dot{\bigcup_{i=1}^{n}} F^{(i)}$$

$$M \in F^{(i)} :\rightsquigarrow F_i \cap M \neq \emptyset \wedge M \cap \bigcup_{j=1}^{i-1} F_j = \emptyset$$

Dann (mit erweiterter Transitionsfunktion $\overline{\delta}$ für NFAs)

$$\overline{\delta}(Q_0, w) \in F^{(i)} \rightsquigarrow w \in [\![\alpha_i]\!] \wedge w \notin \bigcup_{i=1}^{i-1} [\![\alpha_j]\!]$$

Produktive Mengen:

$$M \in P :\rightsquigarrow \exists w \in, \Sigma^* : \overline{\delta}(M,w) \in F \quad \text{(Klar: } F \subseteq P \text{, weil } \overline{\delta}(M,\varepsilon) = M\text{)}$$

Definition des Backtrackautomaten $\mathfrak{B} := \langle Conf, \vdash \rangle$ mit

$$\begin{aligned} Conf &:= (\{N\} \cup \Delta) \times \Sigma^* \mathcal{P}(Q) \Sigma^* \times \Delta^* \{\varepsilon, lexerr\} && \text{Konfigurationenmenge} \\ \vdash &\subseteq Conf^2 && \text{Übergangsrelation} \\ q_0^{\mathfrak{B}} &:= (n, Q_0 w, \varepsilon) && \text{Anfangszustand des BT-Automaten} \end{aligned}$$

normal mode: $M' := \delta(M,a)$

$$(N, Maw, W) \vdash \begin{cases} (N, M'w, W) & \text{falls } M' \in P \setminus F \\ (T_i, M'w, W) & \text{falls } M' \in F^{(i)} \\ \text{Ausgabe: } W \cdot lexerr & \text{falls } M' \notin P \end{cases}$$

backtrack mode:

$$(Q, vMaw, W) \vdash \begin{cases} (T, vaM'w, W) & \text{falls } M' \in P \setminus F \\ (T_i, M'w, W) & \text{falls } M' \in F^{(i)} \\ (N, Q_0 vaw, W) & \text{falls } M' \in P \end{cases}$$

Eingabeende:

$$\begin{aligned} (N, M, W) &\vdash \text{Ausgabe: } W \cdot lexerr && \text{falls } M \in P \setminus F \\ (T, M, W) &\vdash \text{Ausgabe: } WT && \text{falls } M \in F \\ (T, vaM, W) &\vdash (N, Q_0 w, WT) && \text{falls } M \in P \setminus F \end{aligned}$$

Aufgabe 6

```
/* -*- c -*-
 * lexer skeleton for the Universal Toy Scheme Language
 *
 * course:  Compilerbau 2004
 * author:  Michael Weber <michaelw+cb2004@i2.informatik.rwth-aachen.de>
 * file:    $RCSfile: utsl.l,v $
 * date:    $Date: 2004/05/06 12:20:27 $
 * version: $Revision: 1.7.2.1 $
 *
 * Aufgabe: 6
 * Musterloesung
 *
 */

%{
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
```

```
#include <stdio.h>
#include <assert.h>

typedef enum {
  T_EOF, T_LPAREN, T_RPAREN, T_BOOLEAN, T_INTEGER,
  T_FLOAT, T_STRING, T_IDENTIFIER,
} token_t;

typedef struct {
        token_t token;
        union {
                int integer_const;
                int boolean_const;
                double float_const;
                const char *identifier;
                const char *string_const;
        } attrib;
} YYSTYPE;


static token_t new_token (const token_t);
static char *make_identifier (const char *);
static char *make_string_const (const char *);


YYSTYPE yylval;

%}

NEWLINE         \r\n|\n\r|\r|\n
TAB                     \t
SPACE           [ \a\b\f\v]
TOKENSPACE      {NEWLINE}|{TAB}|{SPACE}
COMMENT         ";".*$

LPAREN          "("
RPAREN          ")"
INTEGER         [-+]?[0-9]+
FLOAT           [-+]?([0-9]+[.][0-9]*|[0-9]*[.][0-9]+)([eE][-+]?[0-9]+)?

SPECIAL_INITIAL [-!$%&*/:<=>?^_R]
INITIAL         [[:alpha:]]|{SPECIAL_INITIAL}
SUBSEQUENT      {INITIAL}|[0-9]|{SPECIAL_SUBSEQUENT}
SPECIAL_SUBSEQUENT      [-+.@]
PECULIAR_IDENTIFIER     "+"|"-"|"..."
IDENTIFIER      {INITIAL}{SUBSEQUENT}*|{PECULIAR_IDENTIFIER}
BOOLEAN         "#t"|"#f"
STRING_ELEMENT  [^\\"]|"\\""|"\\\\"
STRING          ["]{STRING_ELEMENT}*["]

%%

{COMMENT}               /* relax */
{TOKENSPACE}    /* relax */
```

```
{LPAREN}                return new_token (T_LPAREN);
{RPAREN}                return new_token (T_RPAREN);


{INTEGER}               {
        new_token (T_INTEGER);
        yylval.attrib.integer_const = atoi(yytext);
        return yylval.token;
}

{FLOAT}         {
        new_token (T_FLOAT);
        yylval.attrib.float_const = atof(yytext);
        return yylval.token;
}


{IDENTIFIER}    {
        new_token (T_IDENTIFIER);
        yylval.attrib.identifier = make_identifier (yytext);
        return yylval.token;
}

{STRING}                {
        new_token (T_STRING);
        yylval.attrib.string_const = make_string_const (yytext);
        return yylval.token;
}

{BOOLEAN}               {
        new_token (T_BOOLEAN);
        yylval.attrib.boolean_const = yytext[1] == 'f' ? 0 : 1;
        return yylval.token;
}

.                               {
        fprintf (stderr, "unexpected character `%c', skipping...\n",
                         yytext[0]);
}

<<EOF>>                 {
        yyterminate();
}

%%

static token_t
new_token (const token_t t)
{
        memset (&yylval, '\0', sizeof yylval);
        yylval.token = t;
        return t;
}
```

```
static char *
make_string_const (const char *yytext)
{
        assert (yytext != NULL);

        return strdup (yytext);
}

static char *
make_identifier (const char *yytext)
{
        assert (yytext != NULL);

        return strdup (yytext);
}

static void
print_symbol (const YYSTYPE symbol)
{
        switch (symbol.token) {
        case T_LPAREN:     printf("(LPAREN . _)"); break;
        case T_RPAREN:     printf("(RPAREN . _)"); break;
        case T_BOOLEAN:    printf("(BOOLEAN . %s)",
                             symbol.attrib.boolean_const == 0 ? "#f" : "#t");
                                                  break;
        case T_INTEGER:    printf("(INTEGER . %d)",
                             symbol.attrib.integer_const); break;
        case T_FLOAT:      printf("(FLOAT . %f)",
                              symbol.attrib.float_const); break;
        case T_STRING:     printf("(STRING . %s)",
                              symbol.attrib.string_const); break;
        case T_IDENTIFIER: printf("(T_IDENTIFIER . %s)",
                              symbol.attrib.identifier); break;
        default:      ;
        }

}

int
main (void)
{
        printf ("(");
        while (yylex() != T_EOF) {
                print_symbol (yylval);
        }
        printf(")\n");

        exit (0);
}
```

Lösungsvorschläge zur 3. Übung „Compilerbau“, SS 2004

Aufgabe 7

Zeige allgemeineren Fall durch Induktion:

$$\alpha \overset{z}{\underset{l}{\Rightarrow}} w \curvearrowright (w,\alpha,y) \overset{*}{\vdash} (\varepsilon,\varepsilon,yz)$$

Sei

$$\begin{aligned}\alpha &= uA\beta\\ w &= uv\\ \pi_i &= A \to \gamma\end{aligned}$$

Fall $z = \varepsilon$:

$$\alpha \overset{l}{\underset{\varepsilon}{\Rightarrow}} w \curvearrowright \alpha = w$$

$$(w,\alpha,y) = (w,w,y) \overset{*}{\vdash} (\varepsilon,\varepsilon,y) \quad \text{(nur Vergleichsschritte)}$$

Fall $z = iz'$:

$$\begin{aligned}\alpha \overset{iz'}{\underset{l}{\Rightarrow}} w &\curvearrowleft uA\beta \overset{iz'}{\underset{l}{\Rightarrow}} uv && \text{(Einsetzen)}\\ &\curvearrowleft uA\beta \overset{i}{\underset{l}{\Rightarrow}} u\gamma\beta \overset{z'}{\underset{l}{\Rightarrow}} uv && \text{(mit } \pi_i\text{)}\end{aligned}$$

Nach Induktionsvoraussetzung gilt $(uv,u\gamma\beta,y') \overset{*}{\vdash} (v,\gamma\beta,y') \overset{*}{\vdash} (\varepsilon,\varepsilon,y'z')$

$$\begin{aligned}(w,\alpha,y) = (uv,uA\beta,y) &\overset{*}{\vdash} (v,A\beta,y)\\ &\vdash (v,\gamma\beta,yi) && \text{mit } y' = yi\\ &= (v,\gamma\beta,y')\\ &\overset{*}{\vdash} (\varepsilon,\varepsilon,y'z')\\ &= (\varepsilon,\varepsilon,yiz')\\ &= (\varepsilon,\varepsilon,yz)\end{aligned}$$

□

Aufgabe 8

a) NTA(G) sei gegeben durch:

- Eingabealphabet: $\Sigma = \{a,b,c,d,e,h,n\}$
- Kelleralphabet: $X = S,A,B,C,D \cup \Sigma$
- Ausgabealphabet: $[12] = 1,\ldots,12$
- Konfigurationenmenge: $\Sigma^* \times X^* \times [p]^*$

Der Automat macht einen

- Ableitungsschritt $(w, A\alpha, z) \vdash (w, \beta\alpha, zi)$ falls $\pi_i = A \to \beta$
- Vergleichsschritt $(aw, a\alpha, z) \vdash (w\alpha, z)$ für $a \in \Sigma$

Die Anfangskonfiguration ist (w, S, ε).

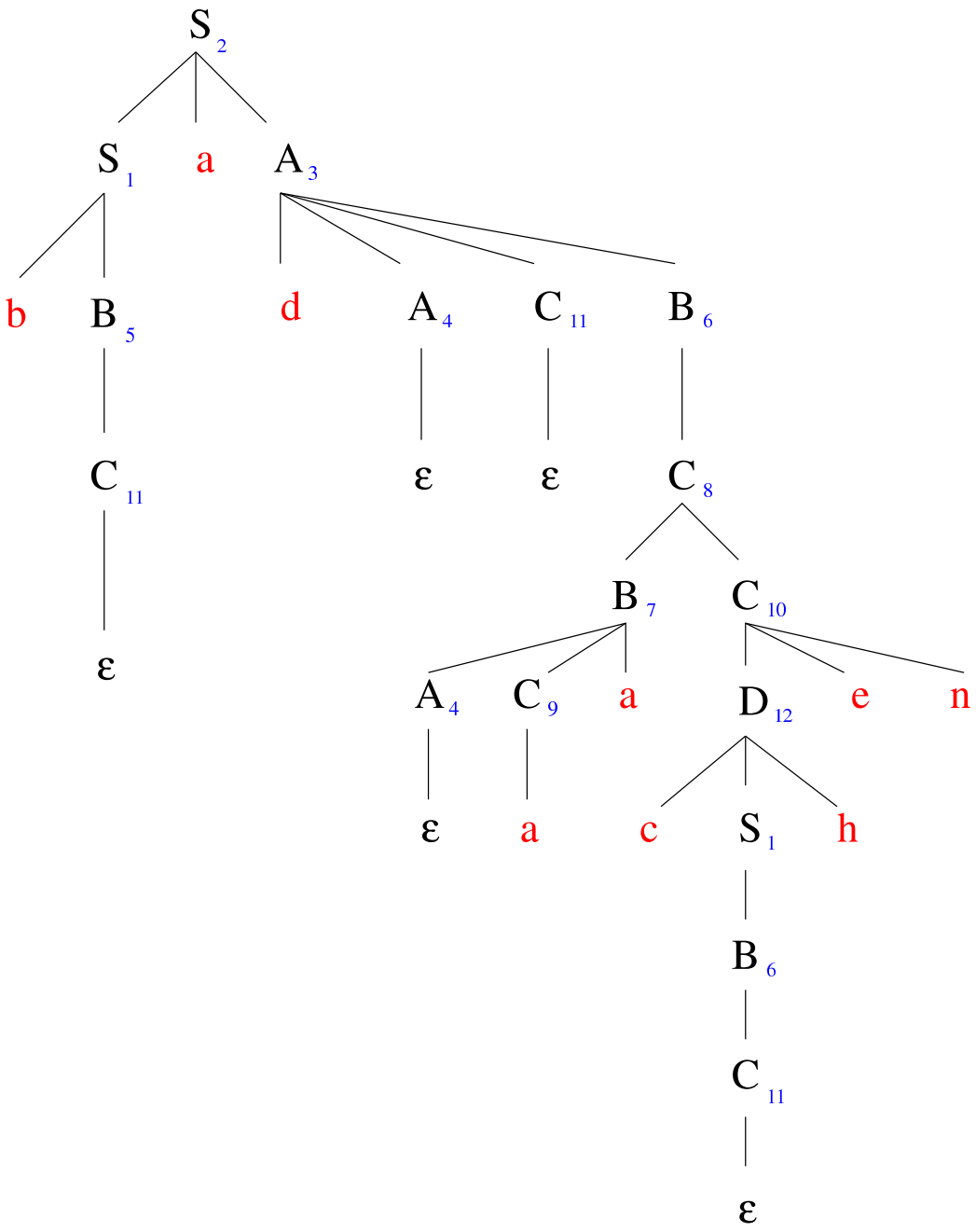
$$
\begin{aligned}
(badaachen, S, \varepsilon) &\vdash (badaachen, SaA, 2) \\
&\vdash (badaachen, BaA, 2\ 1) \\
&\vdash (badaachen, bCaA, 2\ 1\ 5) \\
&\vdash (adaachen, CaA, 2\ 1\ 5) \\
&\vdash (adaachen, aA, 2\ 1\ 5\ 11) \\
&\vdash (daachen, A, 2\ 1\ 5\ 11) \\
&\vdash (daachen, dACB, 2\ 1\ 5\ 11\ 3) \\
&\vdash (aachen, ACB, 2\ 1\ 5\ 11\ 3) \\
&\vdash (aachen, CB, 2\ 1\ 5\ 11\ 3\ 4) \\
&\vdash (aachen, B, 2\ 1\ 5\ 11\ 3\ 4\ 11) \\
&\vdash (aachen, C, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6) \\
&\vdash (aachen, BC, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8) \\
&\vdash (aachen, ACaC, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7) \\
&\vdash (aachen, CaC, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4) \\
&\vdash (aachen, aaC, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9) \\
&\vdash^{*} (chen, C, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9) \\
&\vdash (chen, Den, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10) \\
&\vdash (chen, cShen, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10\ 12) \\
&\vdash (hen, Shen, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10\ 12) \\
&\vdash (hen, Bhen, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10\ 12\ 1) \\
&\vdash (hen, Chen, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10\ 12\ 1\ 6) \\
&\vdash (hen, hen, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10\ 12\ 1\ 6\ 11) \\
&\vdash^{*} (\varepsilon, \varepsilon, 2\ 1\ 5\ 11\ 3\ 4\ 11\ 6\ 8\ 7\ 4\ 9\ 10\ 12\ 1\ 6\ 11)
\end{aligned}
$$

b) Siehe Abbildung 2

c) 1. Ableitungsbaum zur Analyse aufstellen

2. (Preorder-)Rechts-Durchlauf durch den Baum

$z'' = 2 \quad 3 \quad 6 \quad 8 \quad 10 \quad 12 \quad 1 \quad 6 \quad 11 \quad 7 \quad 9 \quad 4 \quad 11 \quad 4 \quad 1 \quad 5 \quad 11$


S 2
S 1
a
A 3
b
B 5
d
A 4
C 11
B 6
C 11
ε
ε
C 8
ε
B 7
C 10
A 4
C 9
a
D 12
e
n
ε
a
c
S 1
h
B 6
C 11
ε

Aufgabe 9

a) $\varepsilon \in \underline{\text{first}}_k(\alpha) \leftrightsquigarrow k = 0$ oder $a \overset{*}{\Rightarrow} \varepsilon$

$$
\begin{array}{rll}
\varepsilon \in \underline{\text{first}}_k(\alpha) & \overset{Def.}{\leftrightsquigarrow} & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} w, |\varepsilon| = k \\
& & \text{oder } \alpha \overset{*}{\Rightarrow} \varepsilon, |\varepsilon| < k \\
& \leftrightsquigarrow & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} w, k = 0 \\
& & \text{oder } \alpha \overset{*}{\Rightarrow} \varepsilon, k > 0 \\
& \overset{G \text{ produktiv}}{\leftrightsquigarrow} & k = 0 \text{ oder } \alpha \overset{*}{\Rightarrow} \varepsilon
\end{array}
$$

□

b) (*) $\alpha \overset{*}{\Rightarrow} \beta \rightsquigarrow \underline{\text{first}}_k(\alpha) \subseteq \underline{\text{first}}_k(\alpha)$

Beweis durch Widerspruch:

Ann.: $\exists v \in \underline{\text{first}}_k(\beta) : v \notin \underline{\text{first}}_k(\alpha)$

$$
\begin{array}{rll}
v \in \underline{\text{first}}_k(\beta) & \overset{Def.}{\leftrightsquigarrow} & \exists w \in \Sigma^* : \beta \overset{*}{\Rightarrow} vw, |v| = k \\
& & \text{oder } \beta \overset{*}{\Rightarrow} v, |v| < k \\
& \overset{Vor.}{\leftrightsquigarrow} & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} \beta \overset{*}{\Rightarrow} vw, |v| = k \\
& & \text{oder } \alpha \overset{*}{\Rightarrow} \beta \overset{*}{\Rightarrow} v, |v| < k \\
& \rightsquigarrow & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} vw, |v| = k \\
& & \text{oder } \alpha \overset{*}{\Rightarrow} v, |v| < k \\
& \rightsquigarrow & v \in \underline{\text{first}}_k(\alpha) \qquad \text{(Widerspruch zur Ann.)}
\end{array}
$$

Also: (*) gilt.

□

c)„$\rightsquigarrow$“:

$$
\begin{array}{rll}
v \in \underline{\text{first}}_k(\alpha) & \rightsquigarrow & \exists x \in \Sigma^* : \alpha \overset{*}{\Rightarrow} x, \{v\} = \underline{\text{first}}_k(x) \\
& \overset{Def.}{\leftrightsquigarrow} & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} vw, |v| = k \\
& & \text{oder } \alpha \overset{*}{\Rightarrow} v, |v| < k \\
& \rightsquigarrow & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} vw \\
& \rightsquigarrow & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} vw, \{v\} = \underline{\text{first}}_k(vw) \\
& \rightsquigarrow & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} x, \{v\} = \underline{\text{first}}_k(x)
\end{array}
$$

„$\leftsquigarrow$“:

$$
\begin{array}{rl}
& \exists x \in \Sigma^* : \alpha \overset{*}{\Rightarrow} x, \{v\} = \underline{\text{first}}_k(x) \rightsquigarrow v \in \underline{\text{first}}_k(\alpha) \\
\overset{x=vw}{\leftsquigarrow} & \exists w \in \Sigma^* : \alpha \overset{*}{\Rightarrow} vw, \{v\} = \underline{\text{first}}_k(vw), |v| = k \\
\rightsquigarrow & v \in \underline{\text{first}}_k(\alpha)
\end{array}
$$

□

Lösungsvorschläge zur 4. Übung „Compilerbau“, SS 2004

Aufgabe 11

a) $G \in \mathrm{LL}(0) \rightsquigarrow |L(G)| = 1$, Die Menge der Sprachen, die genau ein Wort enthalten.

Behauptung: $A \to \beta \mid \gamma \in G \rightsquigarrow \beta = \gamma$

Annahme:

Ex. $A \to \beta \mid \gamma \in G, \beta \neq \gamma$

$$\overset{G \text{ rel.}}{\rightsquigarrow} S \overset{*}{\Rightarrow} wA\alpha \quad \begin{matrix} \overset{l}{\Rightarrow} w\beta\alpha \overset{*}{\underset{l}{\Rightarrow}} wx \\ \overset{l}{\Rightarrow} w\gamma\alpha \overset{*}{\underset{l}{\Rightarrow}} wy \end{matrix}$$

Dann gilt:

$$\underline{\mathrm{first}_0}(x) = \underline{\mathrm{first}_0}(y) = \{\varepsilon\} \quad \rightsquigarrow \quad \beta = \gamma$$

Widerspruch zur Annahme.

$\rightsquigarrow \forall A \to \beta \mid \gamma \in G$ gilt $\beta = \gamma$, da alle Nicht-Terminale produktiv sind.

$\rightsquigarrow |L(G)| = 1$

□

b) 1. $L(\mathrm{RegE}(\Sigma)) = L(\mathrm{DFA}(\Sigma))$
$\mathfrak{A} = \langle Q, \Sigma, \delta, \overline{q}_0, F \rangle \in \mathrm{DFA}(\Sigma)$

2. $G(\mathfrak{A}) = \{A_q \to aA_{q'} \mid \delta(q,a) = q', a \in \Sigma, q \in Q\} \cup \{A_{q_f} \to \varepsilon \mid q_f \in F\}$
Startsymbol: A_{q_0}

3. $G(\mathfrak{A}) in \mathrm{LL}(0)$

$$A_q \to \underbrace{aA_{\delta(q,a)}}_{=:\beta} \mid \underbrace{bA_{\delta(q,b)}}_{=:\gamma}$$

$$a = b \rightsquigarrow \beta = \gamma$$

$\rightsquigarrow \underline{\mathrm{fi}}(\beta) \neq \underline{\mathrm{fi}}(\gamma)$ nach Konstr.

Aufgabe 12

a)

Program	→	`VAR` *id Decl* ; *Stmt*
Decl	→	; *id Decl* | ε
	|	`BEGIN` *Stmt SList* `END`
	|	`IF` *Cond* `THEN` *Stmt*
	|	`WHILE` *Cond* `DO` *Stmt*
SList	→	*Stmt SList* | ε
Cond	→	*Expr ROp Expr*
ROp	→	= | <> | < | >
Expr	→	*id* | *number*
	|	*Expr AOp Expr*
	|	(*Expr*)
AOp	→	+ | −

b)

	$\underline{\text{fi}}$	$\underline{\text{fo}}$
Program	{VAR}	{ε}
Decl	{;, ε}	$\underline{\text{fi}}$(; *Stmt*) = {;}
Stmt	{*id*, BEGIN, IF, WHILE}	$\underline{\text{fo}}$(*Program*) ∪ $\underline{\text{fi}}$(*SList* END) = {*id*, BEGIN, IF, WHILE, END}
SList	{*id*, BEGIN, IF, WHILE, ε}	{END}
Cond	{*id*, *number*, (}	{THEN, DO}
ROp	{=, <>, <, >}	$\underline{\text{fi}}$(*Expr*)
Expr	{*id*, *number*, (}	{;,)} ∪ $\underline{\text{fi}}$(*ROp Expr*) ∪ $\underline{\text{fo}}$(*Cond*) ∪ $\underline{\text{fi}}$(*AOp Expr*) ∪ {)} = {;, =, <>, <, >, THEN, DO, +, −,)}
AOp	{+, −}	$\underline{\text{fi}}$(*Expr*) = {*id*, *number*, (}

c) Zur Erinnerung:
LL(1) : $\underline{\text{la}}(A \to \beta) \cap \underline{\text{la}}(A \to \gamma) \neq \emptyset \rightsquigarrow \beta = \gamma$

$\underline{\text{la}}$(*Stmt* → *id* := *Expr* ;) = {*id*}
$\underline{\text{la}}$(*Stmt* → BEGIN) = {BEGIN}
⋮
$\underline{\text{la}}$(*SList* → *Stmt SList*) = {*id*, BEGIN, *IF*, *WHILE*}
$\underline{\text{la}}$(*SList* → ε) = $\underline{\text{fi}}$(ε$\underline{\text{fo}}$(*SList*)) = {END}
⋮
$\underline{\text{la}}$(*Decl* → ; *id Decl*) = {;}
$\underline{\text{la}}$(*Decl* → ε) = $\underline{\text{fi}}$(ε$\underline{\text{fo}}$(*Decl*)) = {;}
} Schnitt nicht leer, Widerspruch zur LL(1)-Eigenschaft

$\rightsquigarrow G \notin$ LL(1)

d) Linksfaktorisierung (Entscheidung für Regel hinter ; verlagern)

Program → VAR *id* ; *Ids Stmt*
Ids → *id* ; *Ids Stmt* | ε

$\underline{\text{la}}$(*Ids* → *id* ; *Ids*) = {*id*}
$\underline{\text{la}}$(*Ids* → ε) = $\underline{\text{fi}}$(ε$\underline{\text{fo}}$(*Ids*)) = $\underline{\text{fi}}$(*Stmt*) = {*id*, BEGIN, IF, WHILE}
} Widerspr. zur LL(1)-Eigensch.

2. Versuch (hinter *id* verlagern)

Program → VAR *id* ; *Ids*
Ids → $\underline{id}$; *Ids* | $\underline{id}$:= *Expr* ; | $\underbrace{Stmt'}_{\text{ohne } id := Stmt}$

gemeinsames Präfix *id* muß eliminiert werden.

3. Versuch:

Ids → *id I* | *Stmt'*
I → ; *Ids* | :=*Expr* ;

$\underline{\text{la}}(Ids \to id\ I) = \{id\}$
$\underline{\text{la}}(Ids \to Stmt') = \{\texttt{BEGIN}, \texttt{IF}, \texttt{WHILE}\}$
Also: Schnitt leer; weitere Tests: $I, Stmt', \ldots$

Dann: Linksrekursion eliminieren:

$$Expr \quad \to \quad id \mid number \mid Expr\ AOp\ Expr \mid (\ Expr\ )$$

Umformen nach:

$$\begin{array}{lcl} T & \to & id \mid number \mid (\ Expr\ ) \\ Expr & \to & T\ E' \\ E' & \to & AOp\ Expr\ E' \mid \varepsilon \end{array}$$

$\rightsquigarrow G' \in \text{LL}(1)$

e)

	V	id	;	:=	B	E	I	T	W	D	=	<>	<	>	num	+	-	(	)	ε
Prog	1																			
Ids		2			3		3		3											
I			4	5																
Stmt'					6		7		8		16	17	18	19	15	26	26	15	27	
Stmt		9			10		11		12											
SList		13			13	14	13		13											
Cond		15													15					
ROp											16	17	18	19						
Expr		25													25	26	26	25	27	21
E'			27					27			27	27	27	27						
T		22													23			24		
AOp																20	21			

$\texttt{V} = \texttt{VAR}, \texttt{id} = \texttt{ident}, \ldots$

Aufgabe 13

$$
\begin{array}{lcl}
\textit{Program} & \rightarrow & \texttt{(}\ \texttt{var}\ \texttt{(}\ \textit{ident}^{+}\ \texttt{)}\ \textit{Stmt}\ \texttt{)} \\
\textit{Stmt} & \rightarrow & \texttt{(}\ \texttt{setf}\ \textit{ident Expr}\ \texttt{)} \\
 & | & \texttt{(}\ \texttt{begin}\ \textit{Stmt}^{+}\ \texttt{)} \\
 & | & \texttt{(}\ \texttt{if}\ \textit{Cond Stmt}^{+}\ \texttt{)} \\
 & | & \texttt{(}\ \texttt{while}\ \textit{Cond Stmt}^{+}\ \texttt{)} \\
\textit{Cond} & \rightarrow & \texttt{(}\ =\ \textit{Expr Expr}\ \texttt{)} \\
 & | & \texttt{(}\ <>\ \textit{Expr Expr}\ \texttt{)} \\
 & | & \texttt{(}\ <\ \textit{Expr Expr}\ \texttt{)} \\
 & | & \texttt{(}\ >\ \textit{Expr Expr}\ \texttt{)} \\
\textit{Expr} & \rightarrow & \textit{Ident} \\
 & | & \textit{number} \\
 & | & \texttt{(}\ +\ \textit{Expr Expr}\texttt{)} \\
 & | & \texttt{(}\ \texttt{-}\ \textit{Expr Expr}\texttt{)} \\
 & | & \texttt{(}\ \textit{Expr}\ \texttt{)}
\end{array}
$$

Einfachere Grammatik (Rückgewinnung von Informationen durch semantischer Analyse):

$$
\begin{array}{lcl}
\textit{Program} & \rightarrow & \texttt{(}\ \texttt{var}\ \texttt{(}\ \textit{ident}^{+}\ \texttt{)}\ \textit{Stmt}\ \texttt{)} \\
\textit{Stmt} & \rightarrow & \texttt{(}\ \textit{Expr}^{+}\ \texttt{)} \\
\textit{Expr} & \rightarrow & \textit{ident} \\
 & | & \textit{number} \\
 & | & \textit{Stmt}
\end{array}
$$

Lösungsvorschläge zur 5. Übung „Compilerbau“, SS 2004

Aufgabe 14

a)

$$\begin{array}{lcl} S \to S\,S' \mid S & \rightsquigarrow & \begin{array}{lcl} S & \to & S'\,A' \\ A' & \to & S'\,A' \mid \varepsilon \\ S' & \to & a \mid (\,T\,) \end{array} \\ T \to T\,S' \mid \varepsilon & \rightsquigarrow & \begin{array}{lcl} T & \to & A'' \\ A'' & \to & S'\,A'' \mid \varepsilon \end{array} \end{array}$$

LL(1)-Test:

$$\left.\begin{array}{l} \underline{\mathrm{la}}(A' \to S'\,A') = \{a, (\} \\ \underline{\mathrm{la}}(A' \to \varepsilon) = \underline{\mathrm{fi}}(\varepsilon\,\underline{\mathrm{fo}}(A')) = \underline{\mathrm{fo}}(A') = \{\varepsilon\} \end{array}\right\} \text{disjunkt}$$

$$\left.\begin{array}{l} \underline{\mathrm{la}}(S' \to a) = \{a\} \\ \underline{\mathrm{la}}(S' \to (\,T\,)) = \{(\} \end{array}\right\} \text{disjunkt}$$

$$\left.\begin{array}{l} \underline{\mathrm{la}}(A'' \to S'\,A'') = \{a, (\} \\ \underline{\mathrm{la}}(A'' \to \varepsilon) = \underline{\mathrm{fi}}(\varepsilon\,\underline{\mathrm{fo}}(A'')) = \underline{\mathrm{fo}}(A'') = \underline{\mathrm{fo}}(T) = \{)\} \end{array}\right\} \text{disjunkt}$$

$\rightsquigarrow$ Grammatik ist LL(1).

b)

$$\begin{array}{lcl} S \to S'\,S \mid S' & \rightsquigarrow & \begin{array}{lcl} S & \to & S'\,A' \\ A' & \to & S \mid \varepsilon \\ S' & \to & a \mid (\,T'\,) \end{array} \\ T' \to S'\,T' \mid S' & \rightsquigarrow & \begin{array}{lcl} T' & \to & S'\,A'' \\ A'' & \to & T' \mid \varepsilon \end{array} \end{array}$$

c) Lookahead-Mengen:

$$\left.\begin{array}{l} \underline{\mathrm{la}}(A' \to S') = \underline{\mathrm{fi}}(S\,\underline{\mathrm{fo}}(A')) = \underline{\mathrm{fi}}(S') = \{a, (\} \\ \underline{\mathrm{la}}(A' \to \varepsilon) = \underline{\mathrm{fi}}(\varepsilon\,\underline{\mathrm{fo}}(A')) = \underline{\mathrm{fo}}(A') = \underline{\mathrm{fo}}(S) = \{\varepsilon\} \end{array}\right\} \text{disjunkt}$$

$$\left.\begin{array}{l} \underline{\mathrm{la}}(S' \to a) = \ldots \\ \underline{\mathrm{la}}(S' \to (\,T\,)) = \ldots \end{array}\right\} \text{disjunkt}$$

$$\left.\begin{array}{l} \underline{\mathrm{la}}(A'' \to T') = \underline{\mathrm{fi}}(T'\,\underline{\mathrm{fo}}(A'')) = \underline{\mathrm{fi}}(S') = \{a, (\} \\ \underline{\mathrm{la}}(A'' \to \varepsilon) = \underline{\mathrm{fi}}(\varepsilon\,\underline{\mathrm{fo}}(A'')) = \underline{\mathrm{fo}}(A'') = \underline{\mathrm{fo}}(T') = \{)\} \end{array}\right\} \text{disjunkt}$$

$\rightsquigarrow$ Grammatik ist LL(1).

Aufgabe 15

a)

NBA(G)	Σ	Eingabealphabet
	$\mathcal{X}$	Kelleralphabet ($N \cup \Sigma$)
	$[p]$	Ausgabealphabet (Regelnummern)

Konfigurationsmenge $\mathcal{X}^* \times \Sigma^* \times [p]^*$

Transitionen:

shift: $(\alpha, aw, z) \vdash (\alpha a, w, z)$

reduce: $(\beta\alpha, w, z) \vdash (\beta A, w, zi)$ falls $\pi_i = A \to \alpha \in G$

Anfangskonf.: $(\varepsilon, aaba, \varepsilon)$

Endkonf.: (S, ε, z), z ist gespiegelte r-Analyse

Lauf:

$$\begin{aligned}(\varepsilon, aaba, \varepsilon) &\vdash (a, aba, \varepsilon)\\ &\vdash (A, aba, 5)\\ &\vdash (Aa, ba, 5)\\ &\vdash (AA, ba, 55)\\ &\vdash (AAb, a, 55)\\ &\vdash (AS, a, 552)\\ &\vdash (ASa, \varepsilon, 552)\\ &\vdash (ASA, \varepsilon, 5525)\\ &\vdash (AA, \varepsilon, 55254)\\ &\vdash (AAS, \varepsilon, 552543)\\ &\vdash (AS, \varepsilon, 5525431)\\ &\vdash (S, \varepsilon, 55254311)\\ &\vdash (S', \varepsilon, 552543110)\end{aligned}$$

b) Startseparierung $S' \to S$

$$\frac{\text{LR}(0)(\varepsilon)}{\begin{array}{l}S' \to \cdot S\\ S \to \cdot A\,S\\ S \to \cdot A\,b\\ S \to \cdot\\ A \to \cdot SA\\ A \to \cdot a\end{array}} \qquad \frac{\text{LR}(0)(S)}{\begin{array}{l}S' \to S\cdot\\ A \to S\cdot A\\ A \to \cdot SA\\ A \to \cdot a\\ S \to \cdot A\,S\\ S \to \cdot A\,b\\ S \to \cdot\end{array}} \qquad \frac{\text{LR}(0)(SS^+)}{\begin{array}{l}A \to S\cdot A\\ A \to \cdot SA\\ A \to \cdot a\\ S \to \cdot A\,S\\ S \to \cdot A\,b\\ S \to \cdot\end{array}} \qquad \frac{\text{LR}(0)(A^+)}{\begin{array}{l}S \to A\cdot S\\ S \to A\cdot b\\ S \to \cdot A\,S\\ S \to \cdot A\,b\\ S \to \cdot\\ A \to \cdot SA\\ A \to \cdot a\end{array}}$$

$$\frac{\text{LR}(0)(a)}{A \to a\cdot} \qquad \frac{\text{LR}(0)(A^+b)}{S \to A\,b\cdot} \qquad \frac{\text{LR}(0)(A^+S)}{\begin{array}{l}S \to A\,S\cdot\\ A \to S\cdot A\\ A \to \cdot SA\\ A \to \cdot a\\ S \to \cdot A\,S\\ S \to \cdot A\,b\\ S \to \cdot\end{array}} \qquad \frac{\text{LR}(0)(A^+SA)}{\begin{array}{l}A \to SA\cdot\\ S \to \cdot A\cdot S\\ S \to \cdot A\cdot b\\ S \to \cdot A\,S\\ S \to \cdot A\,b\\ S \to \cdot\\ A \to \cdot SA\\ A \to \cdot a\end{array}} \qquad \emptyset$$

$\rightsquigarrow G \notin \text{LR}(0)$ (multiple Konflikte)

Aufgabe 16

siehe Zusatzmaterial

Lösungsvorschläge zur 6. Übung „Compilerbau“, SS 2004

Aufgabe 17

$G \in LL(k) :\leftrightsquigarrow$ für alle Linksableitungen

$$S \overset{*}{\underset{l}{\Rightarrow}} wA\alpha \begin{array}{l} \overset{}{\underset{l}{\Rightarrow}} w\beta\alpha \\ \overset{}{\underset{l}{\Rightarrow}} w\gamma\alpha \end{array} \quad \text{mit } \beta \neq \gamma$$

gilt: $\underline{first}_k(\beta\alpha) \cap \underline{first}_k(\gamma\alpha) = \emptyset$.

$$\begin{aligned} \underline{first}_2(S) &= \{aa, ba, bb\} \\ \underline{first}_2(A) &= \{a, \varepsilon\} \\ \underline{follow}_2(S) &= \{\varepsilon\} \\ \underline{follow}_2(A) &= \{ab, bb\} \end{aligned}$$

$$\underline{la}_2(\pi_1) = \{aa\} \quad (1)$$
$$\underline{la}_2(\pi_2) = \{ba, bb\} \quad (2)$$
$$\underline{la}_2(\pi_3) = \{aa, ab\} \quad (3)$$
$$\underline{la}_2(\pi_4) = \{ab, bb\} \quad (4)$$

$(1) \cap (2) = \emptyset$, aber $(3) \cap (4) \neq \emptyset \Rightarrow G \notin SLL(2)$.
Nun zeigen wir, daß $G \in LL(2)$:

$$S \overset{1}{\underset{l}{\Rightarrow}} aAab \begin{array}{l} \overset{}{\underset{l}{\Rightarrow}} aaab \\ \overset{}{\underset{l}{\Rightarrow}} aab \end{array}$$

$$S \overset{1}{\underset{l}{\Rightarrow}} bAbb \begin{array}{l} \overset{}{\underset{l}{\Rightarrow}} babb \\ \overset{}{\underset{l}{\Rightarrow}} bbb \end{array}$$

Es gilt offensichtlich, dass $\underline{first}_2(aab) \cap \underline{first}_2(ab) = \emptyset$
für den zweiten Fall $\underline{first}_2(abb) \cap \underline{first}_2(bb) = \emptyset$.
Also gilt: $G \in LL(2)$.

Aufgabe 18

a) G_a:

$$\begin{array}{lcll} S & \rightarrow & A & (1) \\ A & \rightarrow & aA \mid \varepsilon & (2,3) \end{array}$$

$G_a \in LL(1)$, da $\underline{la}(\pi_2) \cap \underline{la}(\pi_3) = \{a\} \cap \{\varepsilon\} = \emptyset$.

$LR(0)(\varepsilon)$	$LR(0)(A)$	$LR(0)(a)$	$LR(0)(aA)$	$LR(0)(Aa)$
$S \to \cdot A$ $A \to \cdot aA$ $A \to \cdot$	$S \to A\cdot$	$A \to a \cdot A$ $A \to \cdot aA$ $A \to \cdot$	$A \to aA\cdot$	$\emptyset$

Es treten Shift-Reduce-Konflikte in $LR(0)(\varepsilon)$ und $LR(0)(a)$ auf (die entsprechenden Regeln sind durch **Fettdruck** hervorgehoben). Es folgt daher, daß $G_a \notin LR(0)$. Die Konflikte ließen sich durch die $SLR(1)$-Technik auflösen.

b) G_b: $S \to A \quad (1)$
$A \to aA|ab (2,3)$

$G_b \notin LL(1)$, da $la(\Pi_2) = la(\Pi_3) = \{a\}$.

$LR(0)(\varepsilon)$	$LR(0)(A)$	$LR(0)(a)$	$LR(0)(aA)$	$LR(0)(ab)$	$LR(0)(b)$
$S \to \cdot A$ $A \to \cdot aA$ $A \to \cdot ab$	$S \to A\cdot$	$A \to a \cdot A$ $A \to a \cdot b$ $A \to \cdot aA$ $A \to \cdot ab$	$A \to aA\cdot$	$A \to ab\cdot$	$\emptyset$

Da keine Konflikte in den $LR(0)$-Mengen Auftreten, ist $G_b \in LR(0)$.

c) G_c: $S \to A \quad (1)$
$A \to aA|b \ (2,3)$

$G_c \in LL(1)$, da $la(\Pi_2) \cap la(\Pi_3) = \{a\} \cap \{b\} = \emptyset$.

$LR(0)(\varepsilon)$	$LR(0)(A)$	$LR(0)(a)$	$LR(0)(aA)$	$LR(0)(b)$	$LR(0)(bb)$
$S \to \cdot A$ $S \to \cdot aA$ $S \to \cdot b$	$S \to A\cdot$	$S \to a \cdot A$ $S \to \cdot aA$ $S \to \cdot b$	$S \to aA\cdot$	$S \to b\cdot$	$\emptyset$

Da keine Konflikte in den $LR(0)$-Mengen Auftreten, ist $G_c \in LR(0)$.

Aufgabe 19

a) Füge eine Startseparierung ein:

$$\begin{array}{lcll} S' & \to & S & (0) \\ S & \to & (S)S \mid \varepsilon & (1,2) \end{array}$$

LR(0)(ε)	LR(0)(S)	LR(0)(()	LR(0)((S)	LR(0)((S))	LR(0)((S)S)
S' → .S S → .(S)S S → .	S' → S.	S → (.S)S S → .(S)S S → .	S → (S.)S	S → (S).S S → .(S)S S → .	S → (S)S.

b) Mögliche Konfliktfälle sind zum einen der *shift/reduce*-Konflikt, d.h. eine LR(0)-Menge enthält eine Shift- und eine Reduce-Information (A → $\beta_1 \cdot \beta_2$ und B → $\beta_3\cdot$) und zum anderen der *reduce/reduce*-Konflikt, d.h. eine LR(0)-Menge enthält mehr als eine Reduce-Information. (A → $\beta_1\cdot$ und B → $\beta_3\cdot$)

c) $G(D_1) \notin \text{LR}(0)$, da bereits die Menge LR(0)(ε) einen *shift/reduce*-Konflikt enthält. Betrachte dazu: $S \to \cdot(S)S$ und $S \to \cdot$

Aufgabe 20

a) Abbildung 1

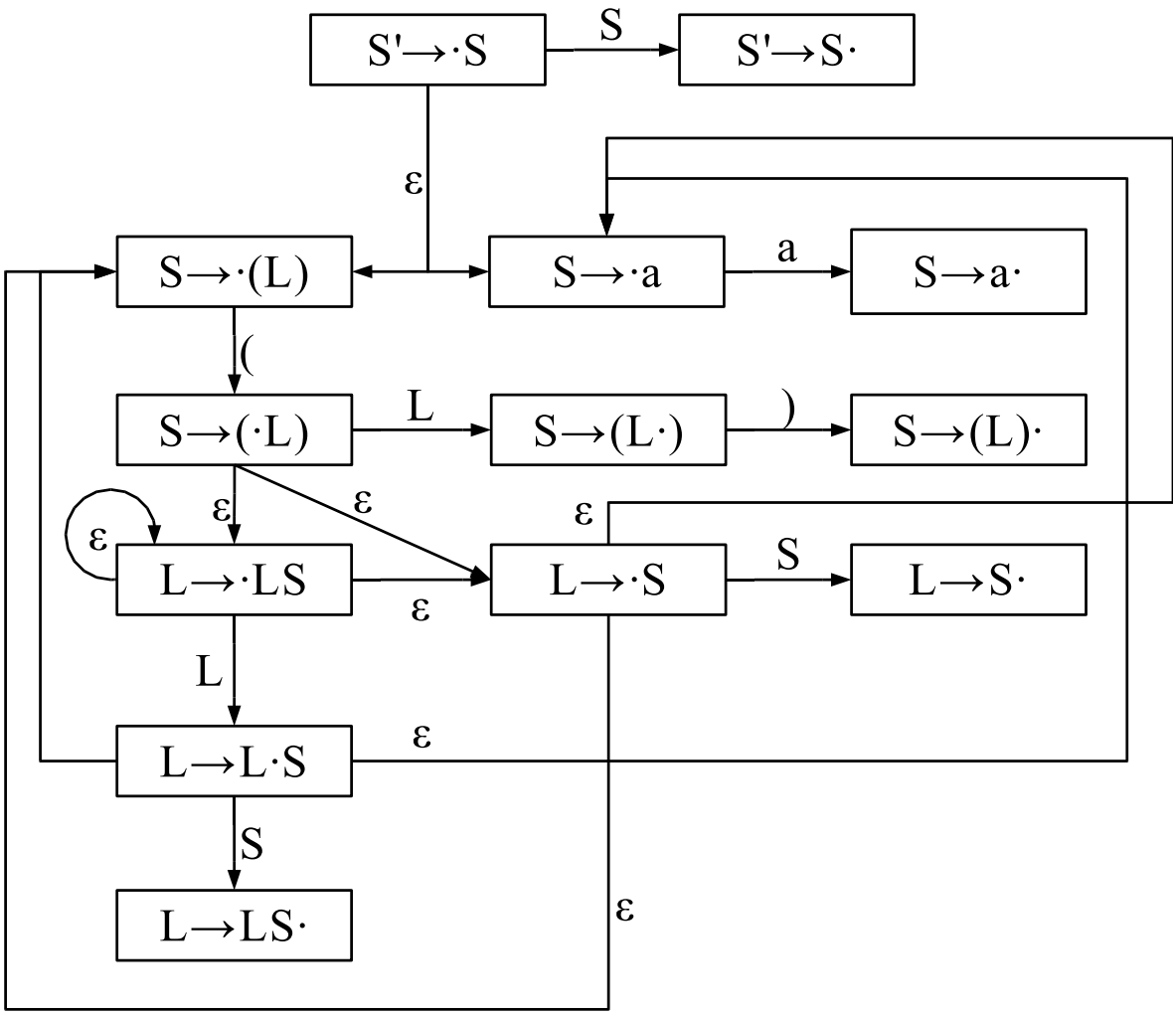


Abbildung 1: Aufgabe 20 (a)

b) Abbildung 2

Achtung: Kante $I_2 \xrightarrow{(} I_2$ fehlt in der Abbildung! In I_0 muß es natürlich $S \to \cdot(L)$ heißen.

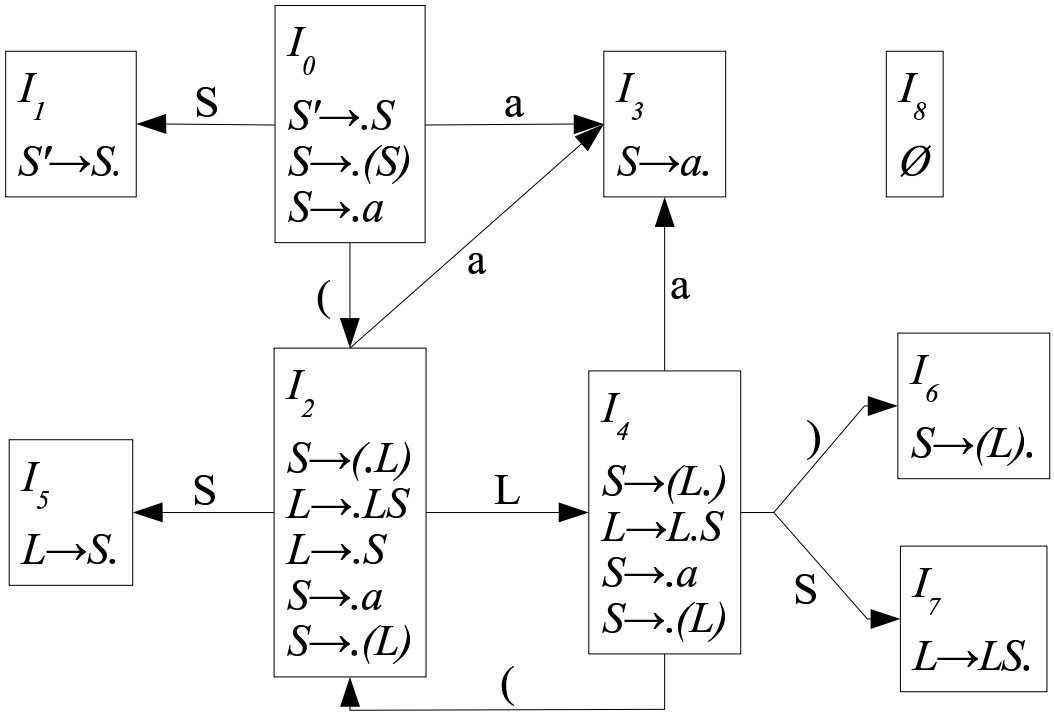


Abbildung 2: Aufgabe 20 (b)

c) *action/goto*-Funktion:

	S	L	(	)	a	action
I_0	I_1		I_2		I_3	shift
I_1						acc
I_2	I_5	I_4	I_2		I_3	shift
I_3						red 2
I_4	I_7		I_2	I_6	I_3	shift
I_5						red 4
I_6						red 1
I_7						red 3
I_8						error

d) $w = (a(aa))$

$$
\begin{aligned}
&(I_0,(a(aa)),\varepsilon) \\
&\vdash (I_0 I_2 I_5,(aa)),2) \\
&\vdash (I_0 I_2 I_4 I_2 I_3,a)),24) \\
&\vdash (I_0 I_2 I_4 I_2 I_4 I_3,)),2424) \\
&\vdash (I_0 I_2 I_4 I_2 I_4 I_6,),242423) \\
&\vdash (I_0 I_2 I_4 I_6,\varepsilon,24242313)
\end{aligned}
$$

$$
\begin{aligned}
&\vdash (I_0 I_2,a(aa)),\varepsilon) \\
&\vdash (I_0 I_2 I_4,(aa)),24) \\
&\vdash (I_0 I_2 I_4 I_2 I_5,a)),242) \\
&\vdash (I_0 I_2 I_4 I_2 I_4 I_7,)),24242) \\
&\vdash (I_0 I_2 I_4 I_7,),2424231) \\
&\vdash (I_0 I_1,\varepsilon,242423131)
\end{aligned}
$$

$$
\begin{aligned}
&\vdash (I_0 I_2 I_3,(aa)),\varepsilon) \\
&\vdash (I_0 I_2 I_4 I_2,aa)),24) \\
&\vdash (I_0 I_2 I_4 I_2 I_4,a)),2424) \\
&\vdash (I_0 I_2 I_4 I_2 I_4,)),242423) \\
&\vdash (I_0 I_2 I_4,),24242313) \\
&\vdash (\varepsilon,\varepsilon,2424231310)
\end{aligned}
$$

Lösungsvorschläge zur 7. Übung „Compilerbau“, SS 2004

Aufgabe 21

Stelle die LR(0)-Mengen auf:

I_0	I_1	I_2	I_3	I_4	I_5	I_6
S’ → .S	S’ → S.	S → A.	S → b.	A → Sb.	A → Aa.	
S → .A	A → S.b	A → A.a				
S → .b						
A → .Aa						
S → .Sb						

Betrachte fo-Mengen:

- $\underline{\text{fo}}(\text{S'}) = \{\varepsilon\}$
- $\underline{\text{fo}}(\text{S}) = \{\varepsilon, b\}$
- $\underline{\text{fo}}(\text{A}) = \{\varepsilon, a, b\}$

Analysetabelle:

action / goto	a	b	$	S	A
I_0	error	shift I_3	error	I_1	I_2
I_1	error	shift I_4	accept		
I_2	shift I_5	red 1	red 1		
I_3	error	red 2	red 2		
I_4	red 4	red 4	red 4		
I_5	red 3	red 3	red 3		
I_6	error	error	error		

Aufgabe 22

a)

$LR(0)(\varepsilon)$	$LR(0)(S)$	$LR(0)(a)$	$LR(0)(ac)$	...
$S' \to \cdot S$	$S' \to S\cdot$	$S \to a \cdot Ad$	$A \to c\cdot$	
$S \to \cdot aAd$		$S \to a \cdot Be$	$B \to c\cdot$	
$S \to \cdot bBd$		$A \to \cdot c$		
$S \to \cdot aBe$		$B \to \cdot c$		
$S \to \cdot bAe$				

In der Menge $LR(0)(ac)$ besteht ein reduce/reduce-Konflikt, der sich nicht durch die SLR(1)-Technik beseitigen läßt, da $\{d, e\} = \underline{\text{fo}}(A) \cap \underline{\text{fo}}(B)$. Also gilt $G \notin SLR(1)$.

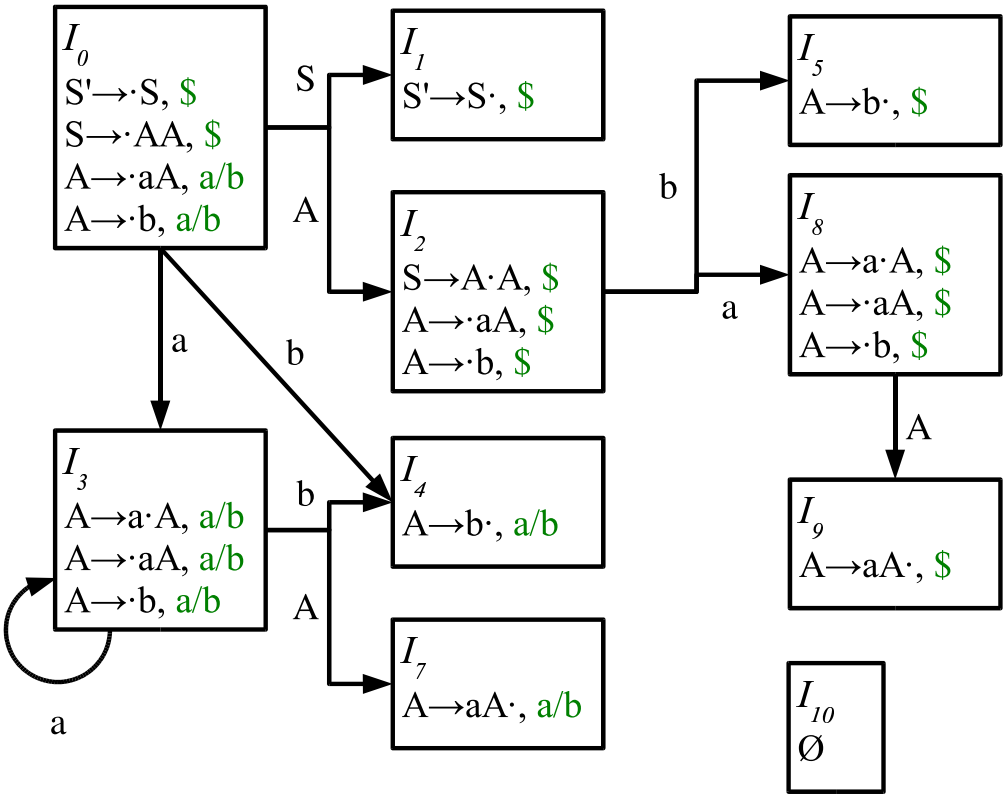


Abbildung 1: Aufgabe 23(a), LALR(1)-Informationen

b)

$LR(1)(\varepsilon)$	$[S' \to \cdot S, \$], [S \to \cdot aAd, \$], [S \to \cdot bBd, \$], [S \to \cdot aBe, \$], [S \to \cdot bAe, \$]$
$LR(1)(S)$	$[S' \to S \cdot, \$]$
$LR(1)(a)$	$[S \to a \cdot Ad, \$], [S \to a \cdot Be, \$], [A \to \cdot c, d], [B \to \cdot c, e]$
$LR(1)(b)$	$[S \to b \cdot Bd, \$], [S \to b \cdot Ae, \$], [B \to \cdot c, d], [A \to \cdot c, e]$
$LR(1)(aA)$	$[S \to aA \cdot d, \$]$
$LR(1)(aB)$	$[S \to aB \cdot e, \$]$
$LR(1)(ac)$	$[A \to c \cdot, d], [B \to c \cdot, e]$
$LR(1)(bA)$	$[S \to bA \cdot e, \$]$
$LR(1)(bB)$	$[S \to bB \cdot d, \$]$
$LR(1)(bc)$	$[B \to c \cdot, d], [A \to c \cdot, e]$
$LR(1)(aAd)$	$[S \to aAd \cdot, \$]$
$LR(1)(aBe)$	$[S \to aBe \cdot, \$]$
$LR(1)(bBd)$	$[S \to bBd \cdot, \$]$
$LR(1)(bAe)$	$[S \to bAe \cdot, \$]$
$LR(1)(d)$	$\emptyset$

Da in den $LR(1)$-Mengen keinerlei Konflikte auftreten, ist $G \in LR(1)$.

Aufgabe 23

a) siehe Abbildung 1 und 2

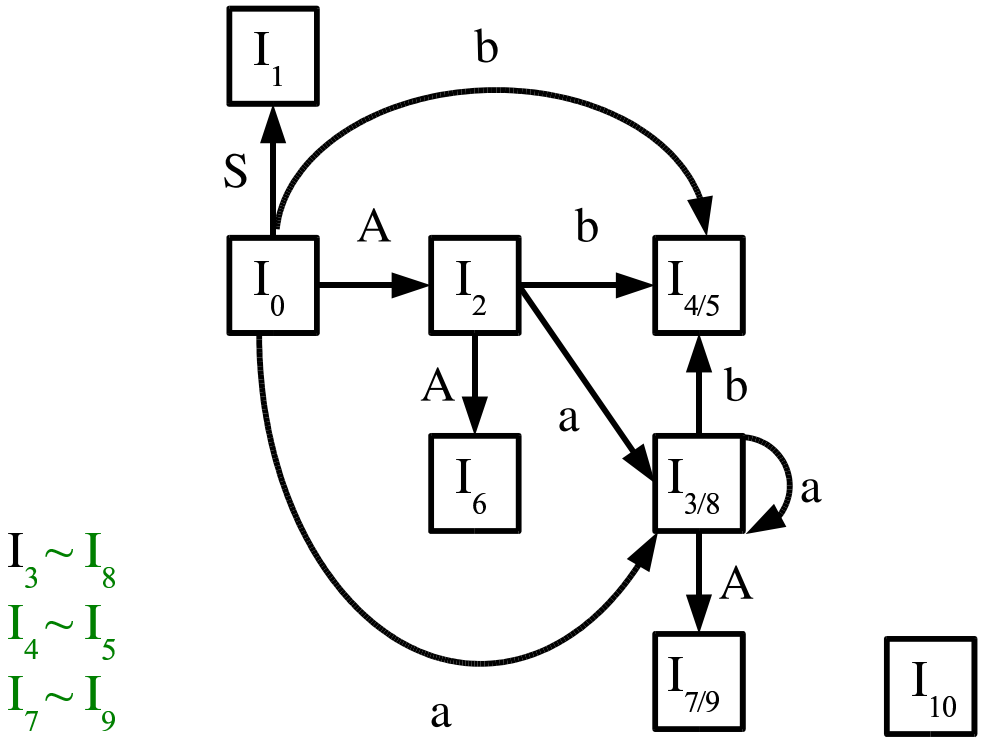


Abbildung 2: Aufgabe 23(a), goto-Funktion

b)

action	*a*	*b*	$
I_0	shift	shift	
I_1			accept
I_2	shift	shift	
$I_{3/8}$	shift	shift	
$I_{4/5}$	reduce 3 (1,A)	reduce 3 (1,A)	reduce 3 (1,A)
I_6			reduce 1 (2,S)
$I_{7/9}$	reduce 2 (2,A)	reduce 2 (2,A)	reduce 2 (2,A)
I_{10}	error	error	error

c)

$$
\begin{array}{lllll}
(I_0, bab, \varepsilon) & \vdash (I_0 I_{4/5} & , ab & , \varepsilon) \\
& \vdash (I_0 I_2 & , ab & , 3) \\
& \vdash (I_0 I_2 I_{3/8} & , b & , 3) \\
& \vdash (I_0 I_2 I_{3/8} I_{4/5} & , \varepsilon & , 3) \\
& \vdash (I_0 I_2 I_{3/8} I_{7/9} & , \varepsilon & , 33) \\
& \vdash (I_0 I_2 I_6 & , \varepsilon & , 332) \\
& \vdash (I_0 I_1 & , \varepsilon & , 3321) \\
& \vdash (\varepsilon & , \varepsilon & , 33210)
\end{array}
$$

Lösungsvorschläge zur 8. Übung „Compilerbau“, SS 2004

Aufgabe 24

a) Bei der Bestimmung der $LALR(1)$-Mengen entsteht unter anderem eine Menge

$$[A \to C\cdot, e]$$
$$[B \to C\cdot, d]$$
$$[A \to C\cdot, d]$$
$$[B \to C\cdot, e]$$

Es besteht offensichtlich ein Reduce-Reduce-Konflikt, da weder mit der aktuellen Reduce-Information, noch mit dem Lookahead bestimmt werden kann, welche Regel angewendet werden soll.

b) Es gelte $I_1 \sim I_2$ für zwei Mengen $I_1, I_2 \in LR(1)$

Fall 1: I_1 und I_2 enthalten nur Shift-Informationen. Dann können in der Vereinigung auch keine Konflikte auftreten, da $I_1 \cup I_2$ auch nur Shift-Informationen enthalten kann.

Fall 2: I_1 enthält nur Shift-Informationen und I_2 enthält sowohl Shift- als auch Reduce-Informationen. Dann wären I_1 und I_2 aber nicht LR(0)-äquivalent.

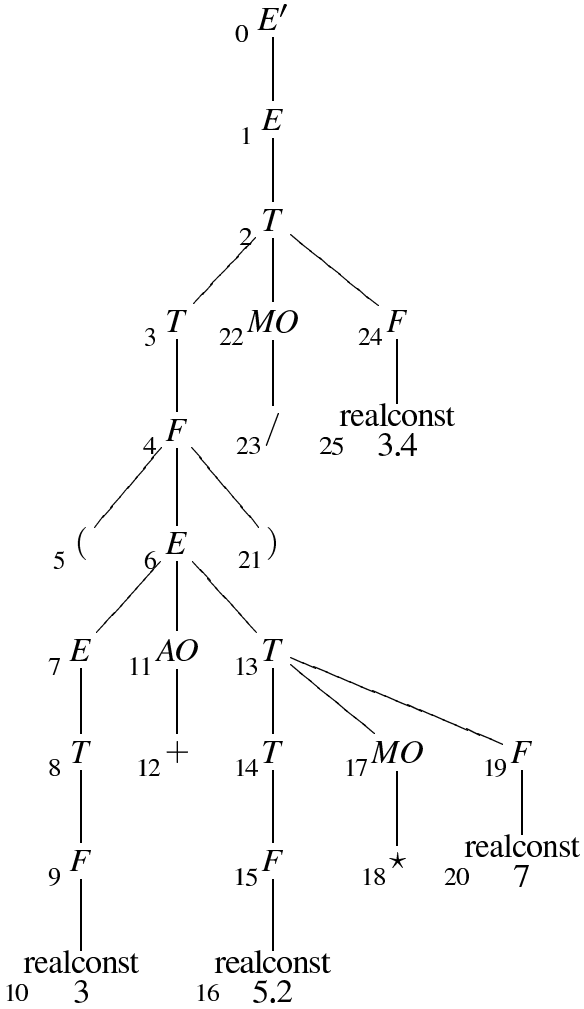
Fall 3: I_1 und I_2 enthalten beide Reduce-Informationen. Dann können Reduce-Reduce-Konflikte auftreten, wie in Aufgabenteil (a) gezeigt wurde.

Aufgabe 25

Attributgrammatik:

F	$\to$ *realconst*	$v.0 = v.1$
F	$\to (E)$	$v.0 = v.1$
MO	$\to *$	$op.0 = *$
MO	$\to /$	$op.0 = /$
T	$\to F$	$v.0 = v.1$
T	$\to T\ MO\ F$	$v.0 = op.2(v.1, v.3)$
AO	$\to +$	$op.0 = +$
AO	$\to -$	$op.0 = -$
E	$\to T$	$v.0 = v.1$
E	$\to E\ AO\ T$	$v.0 = op.2(v.1, v.3)$
E'	$\to E$	$v.0 = v.1$

Ableitungsbaum:



Attributgleichungssystem:

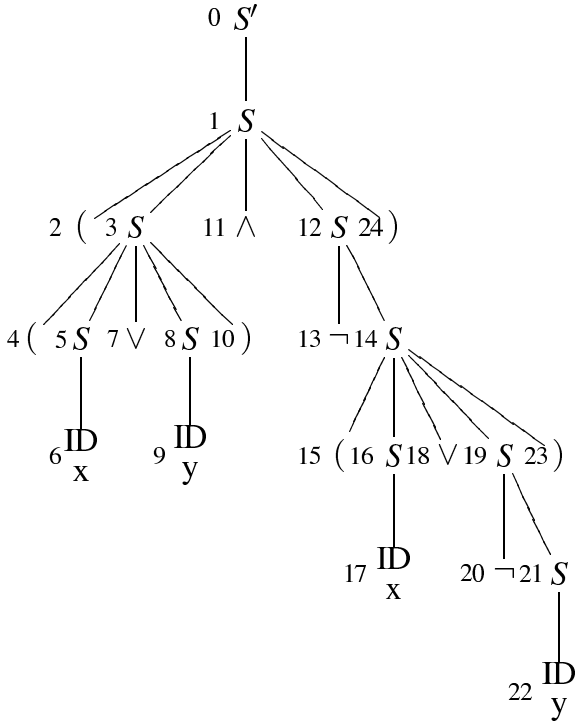
$v.k_0 = v.k_1$	$\approx 11{,}588$
$v.k_1 = v.k_2$	$\approx 11{,}588$
$v.k_2 = op.k_{22}(v.k_3, v.k_{24})$	$= 39{,}4 \div 3{,}4 \approx 11{,}588$
$v.k_3 = v.k_4$	$= 39{,}4$
$v.k_4 = v.k_6$	$= 39{,}4$
$v.k_6 = op.k_{11}(v.k_7, v.k_{13})$	$= 3 + 36{,}4 = 39{,}4$
$v.k_7 = v.k_8$	$= 3$
$v.k_8 = v.k_9$	$= 3$
$v.k_9 = v.k_10$	$= 3$
$v.k_{10} = 3$	
$op.k_{11} = +$	
$v.k_{13} = op.k_{17}(v.k_{14}, v.k_{19})$	$= 5{,}2 \cdot 7 = 36{,}4$
$op.k_{17} = *$	
$v.k_{18} = v.k_{20}$	$= 7$
$v.k_{20} = 7$	
$op.k_{22} = /$	
$v.k_{24} = v.k_{25}$	$= 3{,}4$
$v.k_{25} = 3{,}4$	
$v.k_{14} = v.k_{15}$	$= 5{,}2$
$v.k_{15} = v.k_{16}$	$= 5{,}2$
$v.k_{16} = 5{,}2$	

Aufgabe 26

Attributgrammatik zur Transformation eines vollständig geklammerten Booleschen Ausdrucks in DNF:

$S' \to S$	$n.1 = \neg \circ \neg = id$	
	$\Phi.0 = \Phi.1$	
$S \to (S \vee S)$	$n.1 = n.0$	
	$n.2 = n.0$	
	$\Phi.0 = \begin{cases} \bigcup_{K \in \Phi.1, K' \in \Phi.2} \{K \cup K'\} & \text{falls } n.0 \equiv \neg \\ \Phi.1 \cup \Phi.2 & \text{sonst} \end{cases}$	
$S \to (S \wedge S)$	$n.1 = n.0$	
	$n.2 = n.0$	
	$\Phi.0 = \begin{cases} \Phi.1 \cup \Phi.2 & \text{falls } n.0 \equiv \neg \\ \bigcup_{K \in \Phi.1, K' \in \Phi.2} \{K \cup K'\} & \text{sonst} \end{cases}$	
$S \to \neg S$	$n.1 = n.0 \circ \neg$	
	$\Phi.0 = \Phi.1$	
$S \to ID$	$n.1 = n.0$	
	$\Phi.0 = \{\{n.1(id)\}\}$	

Ableitungsbaum:



Attributgleichungssystem:
Berechnung des inheriten Attributes n:

$$\begin{array}{l|l} n.k_0 = id & \\ n.k_1 = n.k_0 & = id \\ n.k_3 = n.k_1 & = id \\ n.k_{12} = n.k_1 & = id \\ n.k_5 = n.k_3 & = id \\ n.k_8 = n.k_3 & = id \\ n.k_6 = n.k_5 & = id \\ n.k_9 = n.k_8 & = id \\ n.k_{14} = \neg \circ n.k_{12} & = \neg \end{array}$$

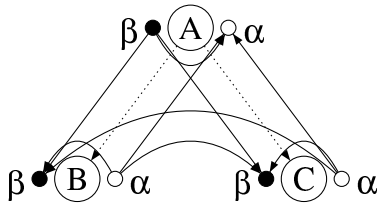
$$
\begin{array}{l|l}
n.k_{16} = n.k_{14} & = \neg \\
n.k_{19} = n.k_{14} & = \neg \\
n.k_{17} = n.k_{16} & = \neg \\
n.k_{21} = \neg \circ n.k_{19} & = id \\
n.k_{22} = n.k_{21} & = id
\end{array}
$$

Berechnung des synthetischen Attributes Φ, welches die Formel in die DNF transformiert (dargestellt als Klauselmenge):

$$
\begin{array}{l|l}
\Phi.0 = \Phi.1 & = \{\{x, \neg x, y\}, \{\neg x, y\}\} \\
\Phi.1 = \begin{cases} \Phi.3 \cup \Phi.12 & \text{falls } n.1 \equiv \neg \\ \bigcup_{K \in \Phi.3, K' \in \Phi.12} \{K \cup K'\} & \text{sonst} \end{cases} & = \{\{x, \neg x, y\}, \{\neg x, y\}\} \\
\Phi.3 = \begin{cases} \bigcup_{K \in \Phi.5, K' \in \Phi.8} \{K \cup K'\} & \text{falls } n.3 \equiv \neg \\ \Phi.5 \cup \Phi.8 & \text{sonst} \end{cases} & = \{\{x\}, \{y\}\} \\
\Phi.5 = \Phi.6 & = \{\{x\}\} \\
\Phi.6 = \{\{n.6(id)\}\} & = \{\{x\}\} \\
\Phi.8 = \Phi.9 & = \{\{y\}\} \\
\Phi.9 = \{\{n.9(id)\}\} & = \{\{y\}\} \\
\Phi.12 = \Phi.14 & = \{\{\neg x, y\}\} \\
\Phi.14 = \begin{cases} \bigcup_{K \in \Phi.16, K' \in \Phi.19} \{K \cup K'\} & \text{falls } n.14 \equiv \neg \\ \Phi.16 \cup \Phi.19 & \text{sonst} \end{cases} & = \{\{\neg x, y\}\} \\
\Phi.16 = \Phi.17 & = \{\{\neg x\}\} \\
\Phi.17 = \{\{n.17(id)\}\} & = \{\{\neg x\}\} \\
\Phi.19 = \Phi.21 & = \{\{y\}\} \\
\Phi.21 = \Phi.22 & = \{\{y\}\} \\
\Phi.22 = \{\{n.22(id)\}\} & = \{\{y\}\}
\end{array}
$$

Lösungsvorschläge zur 9. Übung „Compilerbau“, SS 2004

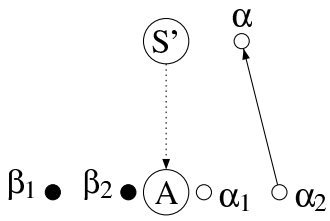
Aufgabe 27



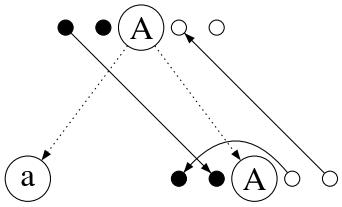
Aufgabe 28

Es ergeben sich folgende Abhängigkeitsgraphen:

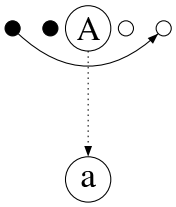
a) DG_{π_1}:



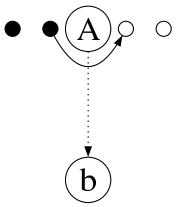
DG_{π_2}:



DG_{π_3}:



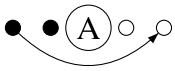
DG_{π_4}:



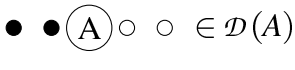
b) Attributabhängigkeitsmengen (grafisch)
(siehe induktive Definition von $\mathcal{D}(A)$)

$D[\pi_3] =$ $\in \mathcal{D}(A)$

$D[\pi_4] =$ $\in \mathcal{D}(A)$

$D[\pi_2;$  $] =$ $\in \mathcal{D}(A)$

$D[\pi_2;$ $] =$ $\in \mathcal{D}(A)$

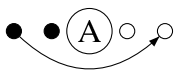
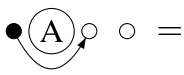
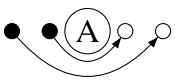
$D[\pi_2;$ $] =$  $\in \mathcal{D}(A)$

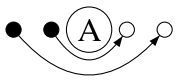
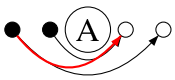
Keine weiteren Elemente in $\mathcal{D}(A)$: fertig.

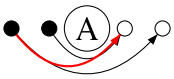
$\mathcal{D}(S) = \{D[\pi_1;D] \mid D \in \mathcal{D}(A)\} = \{$ $\}$

Keine weiteren Elemente in $\mathcal{D}(S)$: fertig.
Wir sehen: die Attributgrammatik G ist nicht zirkulär.

c)

$D(A) \supseteq$  $\cup$  $=$ 

$D(A) \supseteq D[\pi_2;$  $] =$ 

$(\beta_1,\beta_1) \in D[\pi_2;$  $]$, also Zyklus im Abhängigkeitsgraphen.

Folge: G nicht stark-nichtzirkulär

Aufgabe 29

```
%{
#include <stdlib.h>
#include <stdio.h>

#include "utsl-lexer.h"

static int yyerror (const char *);
```

```
static void print_rule (int);
%}

%locations
%error-verbose

%union {
        int integer_const;
        int boolean_const;
        double float_const;
        const char *identifier;
        const char *string_const;
}

%token T_LPAREN
%token T_RPAREN
%token <boolean_const>  T_BOOLEAN
%token <integer_const>  T_INTEGER
%token <float_const>    T_FLOAT
%token <string_const>   T_STRING
%token <identifier>             T_IDENTIFIER

%%

s     : sexpr s1        { print_rule (1); }

s1    : sexpr s1        { print_rule (2); }
      | /*epsilon*/     { print_rule (3); }
          ;

sexpr : atom            { print_rule (4); }
          | T_LPAREN slist T_RPAREN
                        { print_rule (5); }
          ;
```

```
slist : sexpr slist     { print_rule (6); }
          | /*epsilon*/ { print_rule (7); }
          ;

atom  : T_IDENTIFIER    { print_rule (8); }
          | T_BOOLEAN   { print_rule (9); }
          | T_INTEGER   { print_rule (10); }
          | T_FLOAT     { print_rule (11); }
          | T_STRING    { print_rule (12); }
          ;

%%

int
yyerror (const char *s)
{
        fprintf (stderr, "%d:%d:%d:%d: %s\n",
                         yylloc.first_line,
                         yylloc.first_column,
                         yylloc.last_line,
                         yylloc.last_column,
                         s);
        return 1;
}

void
print_rule (int n)
{
        fprintf (stderr, "%d ", n);
}

int
main (void)
```

```
{
        init_location();
        exit (yyparse());
}
```

Lösungsvorschläge zur 10. Übung „Compilerbau“, SS 2004

Aufgabe 30

$$D_{ii} = D[B \to b] = \{(\beta,\alpha)\} \rightsquigarrow \mathcal{D}(B) = \{D_{ii}\}$$
$$D_i = D[A \to a] = \{(\beta,\alpha)\} \rightsquigarrow D_i \in \mathcal{D}(A)$$

$$\begin{aligned} D_{iii} = D[A \to AB; D_i, D_{ii}] &= \{(\alpha,\alpha') \mid (\alpha.0,\alpha'.0) \in trans(Kan(\mathrm{DG}_{A\to AB}) \cup \{(\beta.1,\alpha.1),(\beta.2,\alpha.2)\})\} \\ &= \{(\alpha,\alpha') \mid (\alpha.0,\alpha'.0) \in trans(\{(\alpha.1,\alpha.0),(\beta.0,\beta.2),(\alpha.2,\beta.1), \\ &\qquad (\beta.1,\alpha.1),(\beta.2,\alpha.2)\})\} \end{aligned}$$

wegen Abhängigkeiten $\beta.0 \to \beta.2 \to \alpha.2 \to \beta.1 \to \alpha.1 \to \alpha.0$

$$\begin{aligned} &= \{(\alpha,\alpha') \mid (\alpha.0,\alpha'.0) \in \{(\beta.0,\alpha.0)\}\} \\ &= \{(\beta,\alpha)\} \end{aligned}$$

$D_{iii} \in \mathcal{D}(A)$ bereits enthalten, also keine Änderung, $\mathcal{D}(A)$ fertig

Betrachte nun $\forall D \in \mathcal{D}(A) : D[S \to A; D]$:

$$\begin{aligned} D[S \to A; D_i] &= \{(\alpha,\alpha') \mid (\alpha.0,\alpha'.0) \in trans(Kan(\mathrm{DG}_{S\to A}) \cup \{(\beta.1,\alpha.1)\})\} \\ &= \{(\alpha,\alpha') \mid (\alpha.0,\alpha'.0) \in trans(\{\underline{(\alpha.1,\alpha.1)},(\alpha.1,\beta.1), \\ &\qquad \underline{(\beta.1,\beta.1)},(\beta.1,\alpha.1)\})\} \end{aligned}$$

Es werden nur $(\alpha.\mathbf{0},\alpha'.\mathbf{0})$-Abhängigkeiten berücksichtigt, also

$$= \emptyset$$

Also $\mathcal{D}(S) = \{\emptyset\}$

Zirkularitätstest: $\exists\pi\ldots$
also rate $\pi_1 = S \to A, D_i \in \mathcal{D}(A)$:

$$\begin{aligned} &trans(Kan(\mathrm{DG}_{S\to A}) \cup \{(\alpha.1,\alpha'.1) \mid (\alpha,\alpha') \in D_i\}) \\ &= trans(\{(\alpha.1,\beta.1),(\beta.1,\alpha.1)\}) \ni (\alpha.1,\alpha.1) \end{aligned}$$

Also: Zirkularität gefunden

Aufgabe 31

```
%union {
        /* ... */
        void *ast;
        void *slist;
}

%token T_LPAREN
%token T_RPAREN
%token <boolean_const>  T_BOOLEAN
%token <integer_const>  T_INTEGER
%token <float_const>    T_FLOAT
%token <string_const>   T_STRING
%token <identifier>     T_IDENTIFIER

%type <ast> atom;
%type <ast> sexpr;

%type <slist> slist;
%type <slist> s;

%%

s_    : s           {
        ast_t *a = make_ast_sexpr ($1, &@$); /* we are cheating a litte here... */
        ast_print_dotty (a, "test", stdout);
}

s     : sexpr s         { $$ = slist_push ($1, $2); }
      | sexpr           { $$ = slist_push ($1, make_slist_alloc()); }

sexpr : atom            { $$ = $1; }
          | T_LPAREN slist T_RPAREN
                        { $$ = make_ast_sexpr ($2, &@$); }
          ;

slist : sexpr slist     { $$ = slist_push ($1, $2); }
      | /*epsilon*/     { $$ = make_slist_alloc(); }
          ;

atom  : T_IDENTIFIER{ $$ = make_ast_identifier ($1, &@$); }
          | T_BOOLEAN   { $$ = make_ast_boolean_const ($1, &@$); }
          | T_INTEGER   { $$ = make_ast_integer_const ($1, &@$); }
          | T_FLOAT     { $$ = make_ast_float_const ($1, &@$); }
          | T_STRING    { $$ = make_ast_string_const ($1, &@$); }
          ;
%%
```

Lösungsvorschläge zur 11. Übung „Compilerbau“, SS 2004

Aufgabe 32

a) Attributschema zur Typberechnung:

$SExpr \rightarrow \texttt{nil}$

$type.0 = \texttt{null}$

$SExpr \rightarrow \texttt{(setf}\ identifier\ SExpr\texttt{)}$

$symtab.3 = symtab.4 = symtab.0$

$$type.0 = \begin{cases} \texttt{null} & \text{falls } symtab.0(ident.3) = type.4 \in T \setminus \{\texttt{error}\} \\ \texttt{error} & \text{sonst} \end{cases}$$

$SExpr \rightarrow \texttt{(if}\ SExpr\ SExpr\ SExpr\texttt{)}$

$symtab.3 = symtab.4 = symtab.5 = symtab.0$

$$type.0 = \begin{cases} type.4 & \text{falls } type.3 = \texttt{boolean}, type.4 = type.5 \\ \texttt{error} & \text{sonst} \end{cases}$$

$SExpr \rightarrow \texttt{(while}\ SExpr\ SList\texttt{)}$

$symtab.3 = symtab.4 = symtab.0$

$$type.0 = \begin{cases} \texttt{null} & \text{falls } type.3 = \texttt{boolean}, type.4 \neq \texttt{error} \\ \texttt{error} & \text{sonst} \end{cases}$$

$SExpr \rightarrow \texttt{(let (}LetDefs\texttt{)}\ SList\texttt{)}$

$symtab.4 = symtab.0$

$symtab.6 = tab.4 \oplus symtab.0$

$$type.0 = \begin{cases} t_m & \text{falls } type.6 = t_1 \times \cdots \times t_m\ \forall t_i \in T \setminus \{\texttt{error}\} \text{ und } type.4 \neq \texttt{error} \\ \texttt{error} & \text{sonst} \end{cases}$$

$SExpr \rightarrow \texttt{(}SList\texttt{)}$

$symtab.2 = symtab.0$

$$type.0 = \begin{cases} t_r & \text{falls } type.2 = t_f \times t_1 \times \cdots \times t_m, \\ & \qquad t_f = t_1 \times \cdots \times t_n \rightarrow t_r \\ & \qquad t_f, t_i \in T \setminus \{\texttt{error}\} \\ \texttt{error} & \text{sonst} \end{cases}$$

$SExpr \rightarrow Atom$

$symtab.1 = symtab.0$

$type.0 = type.1$

$SList \rightarrow SExpr\ SList \mid \varepsilon$

$symtab.1 = symtab.2 = symtab.0$

$$type.0 = \begin{cases} type.1 \times type.2 & \text{falls } type.1, type.2 \neq \texttt{error} \\ \texttt{error} & \text{sonst} \end{cases}$$

$$
\begin{aligned}
&\mathit{LetDefs} \rightarrow (\mathit{identifier}\ \mathit{SExpr})\ \mathit{LetDefs}\\
&\qquad \mathit{symtab}.2 = \mathit{symtab}.3 = \mathit{symtab}.4 = \mathit{symtab}.0\\
&\qquad \mathit{tab}.0 = \mathit{tab}.5[\mathit{ident}.2/\mathit{type}.3]\\
&\qquad \mathit{type}.0 = \begin{cases} \texttt{lval}\ \mathit{type}.3 & \text{falls } \mathit{type}.3, \mathit{type}.5 \neq \texttt{error}\\ \texttt{error} & \text{sonst} \end{cases}\\
&\mathit{LetDefs} \rightarrow \varepsilon\\
&\qquad \mathit{tab}.0 = []\\
&\qquad \mathit{type}.0 = \bot\\
&\mathit{Atom} \rightarrow \mathit{identifier}\\
&\qquad \mathit{type}.0 = \mathit{symtab}.0(\mathit{ident}.1)\\
&\mathit{Atom} \rightarrow \mathit{boolean}\\
&\qquad \mathit{type}.0 = \texttt{boolean}\\
&\mathit{Atom} \rightarrow \mathit{integer}\\
&\qquad \mathit{type}.0 = \texttt{integer}\\
&\mathit{Atom} \rightarrow \mathit{float}\\
&\qquad \mathit{type}.0 = \texttt{float}
\end{aligned}
$$

b) Abbildung 1 zeigt den Ableitungsbaum mit ausgewertetem *type*-Attribut.
Das Attributgleichungssystem:

```
[00254,i] symtab.2 = symtab.1 = [*/fun, +/fun, </fun, =/fun, ]
[00267,i] symtab.3 = symtab.1 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.4 = symtab.3 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.5 = symtab.3 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.8 = symtab.3 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.6 = symtab.5 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.7 = symtab.6 = [*/fun, +/fun, </fun, =/fun, ]
[00284,s] type.7 = int
[00429,s] type.6 = type.7 = int
[00421,s] type.5 = type.6 = int
[00254,i] symtab.9 = symtab.8 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.10 = symtab.8 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.13 = symtab.8 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.11 = symtab.10 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.12 = symtab.11 = [*/fun, +/fun, </fun, =/fun, ]
[00284,s] type.12 = int
[00429,s] type.11 = type.12 = int
[00421,s] type.10 = type.11 = int
[00254,i] symtab.14 = symtab.13 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.15 = symtab.13 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.16 = symtab.15 = [*/fun, +/fun, </fun, =/fun, ]
[00254,i] symtab.17 = symtab.16 = [*/fun, +/fun, </fun, =/fun, ]
[00284,s] type.17 = int
[00429,s] type.16 = type.17 = int
[00421,s] type.15 = type.16 = int
[00317,s] tab.13 = [id.14=z/type.15] = [z/int, ]
```

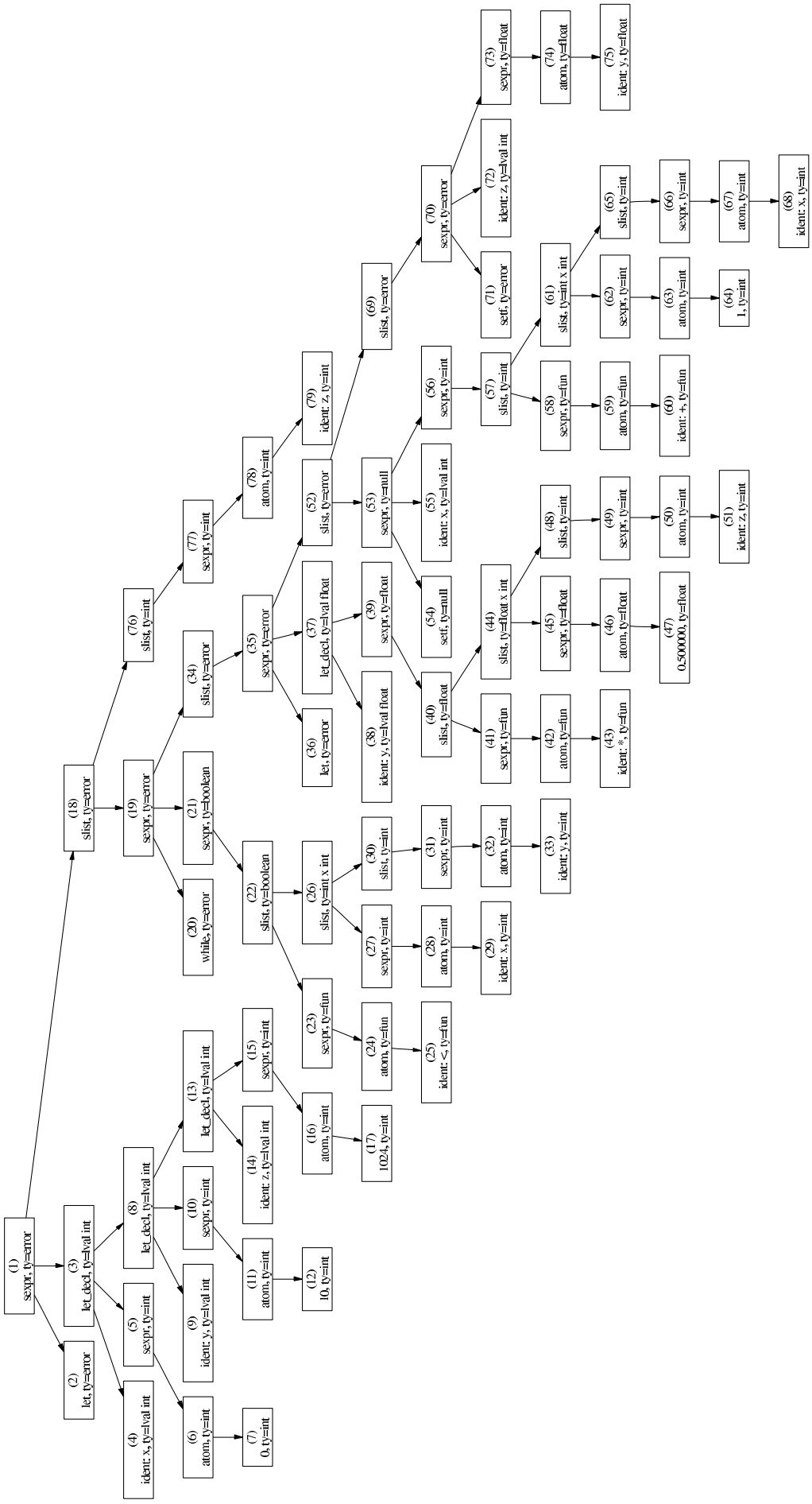


Abbildung 1: Ableitungsbaum

```
[00324,s] type.14 = lval type.15 = lval int
[00343,s] type.13 = type.14 = lval int
[00314,s] tab.8 = tab.13[id.9=y/type.10] = [y/int, z/int, ]
[00324,s] type.9 = lval type.10 = lval int
[00337,s] type.8 = type.9 = lval int
[00314,s] tab.3 = tab.8[id.4=x/type.5] = [x/int, y/int, z/int, ]
[00324,s] type.4 = lval type.5 = lval int
[00337,s] type.3 = type.4 = lval int
[00357,i] symtab.18 = tab.3 (+) symtab.1 = [x/int, y/int, z/int, */fun, +/fun, </fun
[00254,i] symtab.19 = symtab.18 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.76 = symtab.18 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.20 = symtab.19 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.21 = symtab.20 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.34 = symtab.20 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.22 = symtab.21 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.23 = symtab.22 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.26 = symtab.22 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.24 = symtab.23 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.25 = symtab.24 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00442,s] type.25 = symtab.25(id.25=<) = fun
[00429,s] type.24 = type.25 = fun
[00421,s] type.23 = type.24 = fun
[00254,i] symtab.27 = symtab.26 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.30 = symtab.26 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.28 = symtab.27 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.29 = symtab.28 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00442,s] type.29 = symtab.29(id.29=x) = int
[00429,s] type.28 = type.29 = int
[00421,s] type.27 = type.28 = int
[00254,i] symtab.31 = symtab.30 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.32 = symtab.31 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.33 = symtab.32 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00442,s] type.33 = symtab.33(id.33=y) = int
[00429,s] type.32 = type.33 = int
[00421,s] type.31 = type.32 = int
[00409,s] type.30 = type.31 = int
[00405,s] type.26 = type.27 x type.30 = int x int
[00396,s] type.22 = type_check_fun(<, type.26) = boolean
[00421,s] type.21 = type.22 = boolean
[...]
[00442,s] type.75 = symtab.75(id.75=y) = float
[00429,s] type.74 = type.75 = float
[00421,s] type.73 = type.74 = float
[00457,s] type.71 = type_check_setf(symtab.71(id.72=z), type.73) = error
[00421,s] type.70 = type.71 = error
[00409,s] type.69 = type.70 = error
[00405,s] type.52 = type.53 x type.69 = error
[00381,s] type.36 = type.52 = error
[00421,s] type.35 = type.36 = error
[00409,s] type.34 = type.35 = error
[00470,s] type.20 = type_check_while(type.21, type.34) = error
```

```
[00421,s] type.19 = type.20 = error
[00254,i] symtab.77 = symtab.76 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.78 = symtab.77 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00254,i] symtab.79 = symtab.78 = [x/int, y/int, z/int, */fun, +/fun, </fun, =/fun,
[00442,s] type.79 = symtab.79(id.79=z) = int
[00429,s] type.78 = type.79 = int
[00421,s] type.77 = type.78 = int
[00409,s] type.76 = type.77 = int
[00405,s] type.18 = type.19 x type.76 = error
[00381,s] type.2 = type.18 = error
[00421,s] type.1 = type.2 = error
```

Lösungsvorschläge zur 12. Übung „Compilerbau“, SS 2004

Aufgabe 33

$$
\begin{aligned}
\mathfrak{M}[\![P]\!](1) &= M[\![\textbf{in/out } Y; \textbf{const } X = 4; \textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1.]\!](1) \\
&= (\mathfrak{B}[\![\textbf{const } X = 4; \textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1.]\!] \underbrace{\rho_\emptyset[Y/\alpha_1]}_{=:\rho_1} \underbrace{\sigma_\emptyset[\alpha_1/1]}_{=:\sigma_1}) \\
&= (\mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1.]\!](\mathfrak{D}[\![\textbf{const } X = 4;]\!])\rho_1\sigma_1) \\
&= (\mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1.]\!] \underbrace{\rho_1[X/4]}_{=:\rho_2} \sigma_1) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1]\!]\rho_2(\mathfrak{C}[\![Y := Y + 1]\!]\rho_2\sigma_1) & \text{, falls } \mathfrak{E}_b[\![(Y * Y) < X]\!]\rho_2\sigma_1 = \textbf{true} \\ \sigma_1 & \text{, falls } \mathfrak{E}_b[\![(Y * Y) < X]\!]\rho_2\sigma_1 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1]\!]\rho_2(\mathfrak{C}[\![Y := Y + 1]\!]\rho_2\sigma_1) & \text{, falls } \mathfrak{E}_a[\![Y * Y]\!]\rho_2\sigma_1 < \mathfrak{E}_a[\![X]\!]\rho_2\sigma_1 = \textbf{true} \\ \sigma_1 & \text{, falls } \mathfrak{E}_a[\![Y * Y]\!]\rho_2\sigma_1 < \mathfrak{E}_a[\![X]\!]\rho_2\sigma_1 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1]\!]\rho_2(\mathfrak{C}[\![Y := Y + 1]\!]\rho_2\sigma_1) & \text{, falls } \mathfrak{E}_a[\![Y]\!]\rho_2\sigma_1 \cdot \mathfrak{E}_a[\![Y]\!]\rho_2\sigma_1 < 4 = \textbf{true} \\ \sigma_1 & \text{, falls } \mathfrak{E}_a[\![Y]\!]\rho_2\sigma_1 \cdot \mathfrak{E}_a[\![Y]\!]\rho_2\sigma_1 < 4 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1]\!]\rho_2(\mathfrak{C}[\![Y := Y + 1]\!]\rho_2\sigma_1) & \text{, falls } \sigma(\alpha_1) \cdot \sigma(\alpha_1) < 4 = \textbf{true} \\ \sigma_1 & \text{, falls } \sigma(\alpha_1) \cdot \sigma(\alpha_1) < 4 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1]\!]\rho_2(\mathfrak{C}[\![Y := Y + 1]\!]\rho_2\sigma_1) & \text{, falls } 1 \cdot 1 < 4 = \textbf{true} \\ \sigma_1 & \text{, falls } 1 \cdot 1 < 4 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y * Y) < X \textbf{ do } Y := Y + 1]\!]\rho_2(\mathfrak{C}[\![Y := Y + 1]\!]\rho_2\sigma_1) & \text{, falls } \textbf{true} = \textbf{true} \\ \sigma_1 & \text{, falls } \textbf{true} = \textbf{false} \end{cases} \right)
\end{aligned}
$$

$$
\begin{aligned}
&= (\mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2(\mathfrak{C}[\![Y := Y+1]\!]\rho_2\sigma_1)) \\
&= (\mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2\sigma_1[\rho_2(Y)/\mathfrak{E}_a[\![Y+1]\!]\rho_2\sigma_1]) \\
&= (\mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2\sigma_1[\alpha_1/(\mathfrak{E}_a[\![Y]\!]\rho_2\sigma_1 + \mathfrak{E}_a[\![1]\!]\rho_2\sigma_1)]) \\
&= (\mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2\sigma_1[\alpha_1/(\sigma_1(\alpha_1)+1)]) \\
&= (\mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2 \underbrace{\sigma_1[\alpha_1/2]}_{=:\sigma_2}) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2(\mathfrak{C}[\![Y := Y+1]\!]\rho_2\sigma_2) & \text{, falls } \mathfrak{E}_b[\![(Y*Y) < X]\!]\rho_2\sigma_2 = \textbf{true} \\ \sigma_2 & \text{, falls } \mathfrak{E}_b[\![(Y*Y) < X]\!]\rho_2\sigma_2 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2(\mathfrak{C}[\![Y := Y+1]\!]\rho_2\sigma_2) & \text{, falls } 2\cdot 2 < 4 = \textbf{true} \\ \sigma_2 & \text{, falls } 2\cdot 2 < 4 = \textbf{false} \end{cases} \right) \\
&= \left(\begin{cases} \mathfrak{C}[\![\textbf{while } (Y*Y) < X \textbf{ do } Y := Y+1]\!]\rho_2(\mathfrak{C}[\![Y := Y+1]\!]\rho_2\sigma_2) & \text{, falls } \textbf{false} = \textbf{true} \\ \sigma_2 & \text{, falls } \textbf{false} = \textbf{false} \end{cases} \right) \\
&= (\sigma_2) \\
&= (\{\alpha_1 \mapsto 2\})
\end{aligned}
$$

Aufgabe 34

```
in/out N, FN;
    var F0, F1;
    F0 := 0;
    F1 := 1;
    FN := F0;
    while (0 < N) do
    begin
      FN := F1;
      F1 := F0 + F1;
      F0 := FN;
      N := N - 1;
    end
end.
```

Aufgabe 35

Iterativ:

```
 1: CALL(3,0,2)     ; Call Main
 2: JMP(0)          ; End
 3: LIT(0)          ; Begin Main
 4: STORE(0,1)      ; F0 := 0
 5: LIT(1)
 6: STORE(0,2)      ; F1 := 1
 7: LOAD(0,1)
 8: STORE(1,2)      ; FN := F0
 9: LIT(0)
10: LOAD(1,1)
11: LT              ; 0 < N
12: JFALSE(26)      ; while
13: LOAD(0,2)
14: STORE(1,2)      ; FN := F1
15: LOAD(0,1)
16: LOAD(0,2)
17: ADD
18: STORE(0,2)      ; F1 := F0 + F1
19: LOAD(1,2)
20: STORE(0,1)      ; F0 := FN
21: LOAD(1,1)
22: LIT(-1)
23: ADD
24: STORE(1,1)      ; N := N - 1
25: JMP(9)          ; End while
26: RET             ; End Main
```

```
 Datenkeller:
 Prozedurkeller: 0 0 0 1 42
Codezeile 001   CALL(3,0,2)     ; Call Main
 Datenkeller:
 Prozedurkeller: 5 4 2 0 0 0 0 0 1 42
Codezeile 003   LIT(0)          ; Begin Main
 Datenkeller: 0
 Prozedurkeller: 5 4 2 0 0 0 0 0 1 42
Codezeile 004   STORE(0,1)      ; F0 := 0
 Datenkeller:
 Prozedurkeller: 5 4 2 0 0 0 0 0 1 42
Codezeile 005   LIT(1)
 Datenkeller: 1
 Prozedurkeller: 5 4 2 0 0 0 0 0 1 42
Codezeile 006   STORE(0,2)      ; F1 := 1
 Datenkeller:
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 42
Codezeile 007   LOAD(0,1)
 Datenkeller: 0
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 42
Codezeile 008   STORE(1,2)      ; FN := F0
 Datenkeller:
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 0
Codezeile 009   LIT(0)
 Datenkeller: 0
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 0
Codezeile 010   LOAD(1,1)
 Datenkeller: 0 1
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 0
Codezeile 011   LT              ; 0 < N
 Datenkeller: 1
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 0
Codezeile 012   JFALSE(26)      ; while
 Datenkeller:
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 0
Codezeile 013   LOAD(0,2)
 Datenkeller: 1
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 0
Codezeile 014   STORE(1,2)      ; FN := F1
 Datenkeller:
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 1
```

```
Codezeile 015   LOAD(0,1)
 Datenkeller: 0
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 1
Codezeile 016   LOAD(0,2)
 Datenkeller: 0 1
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 1
Codezeile 017   ADD
 Datenkeller: 1
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 1
Codezeile 018   STORE(0,2)      ; F1 := F0 + F1
 Datenkeller:
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 1
Codezeile 019   LOAD(1,2)
 Datenkeller: 1
 Prozedurkeller: 5 4 2 0 1 0 0 0 1 1
Codezeile 020   STORE(0,1)      ; F0 := FN
 Datenkeller:
 Prozedurkeller: 5 4 2 1 1 0 0 0 1 1
Codezeile 021   LOAD(1,1)
 Datenkeller: 1
 Prozedurkeller: 5 4 2 1 1 0 0 0 1 1
Codezeile 022   LIT(-1)
 Datenkeller: 1 -1
 Prozedurkeller: 5 4 2 1 1 0 0 0 1 1
Codezeile 023   ADD
 Datenkeller: 0
 Prozedurkeller: 5 4 2 1 1 0 0 0 1 1
Codezeile 024   STORE(1,1)      ; N := N - 1
 Datenkeller:
 Prozedurkeller: 5 4 2 1 1 0 0 0 0 1
Codezeile 025   JMP(9)          ; End while
 Datenkeller:
 Prozedurkeller: 5 4 2 1 1 0 0 0 0 1
Codezeile 009   LIT(0)
 Datenkeller: 0
 Prozedurkeller: 5 4 2 1 1 0 0 0 0 1
Codezeile 010   LOAD(1,1)
 Datenkeller: 0 0
 Prozedurkeller: 5 4 2 1 1 0 0 0 0 1
Codezeile 011   LT              ; 0 < N
 Datenkeller: 0
```

```
 Prozedurkeller: 5 4 2 1 1 0 0 0 0 1
Codezeile 012   JFALSE(26)      ; while
 Datenkeller:
 Prozedurkeller: 5 4 2 1 1 0 0 0 0 1
Codezeile 026   RET             ; End Main
 Datenkeller:
 Prozedurkeller: 0 0 0 0 1
Codezeile 002   JMP(0)          ; End
 Datenkeller:
 Prozedurkeller: 0 0 0 0 1

I/O-Variable 1 = 0
I/O-Variable 2 = 1
```

Rekursiv:

```
 1: LOAD(0,1)          ; I/O-Variable auf Datenkeller laden
 2: CALL(5,0,1)        ; Main-Block aufrufen, Prozedurparameter auf Da
 3: STORE(0,1)         ; Ergebnis speichern
 4: JMP(0)
 5: STORE(0,1)         ; Prozedurparameter in lokaler Variablen sicher
 6: LOAD(0,1)          ; und wieder auf den Keller schreiben
 7: LIT(1)
 8: LT                 ; N<=0 ?
 9: JFALSE(12)
10: LIT(0)
11: RET                ; falls ja: 0 zurueckgeben
12: LOAD(0,1)
13: LIT(2)
14: LT                 ; N=1 ?
15: JFALSE(18)
16: LIT(1)
17: RET                ; falls ja: 1 zurueckgeben
18: LOAD(0,1)
19: LIT(-1)
20: ADD                ; N-1 auf Datenkeller
21: CALL(5,1,1)        ; Rekursiver Aufruf f(N-1) mit Parameter auf Da
22: LOAD(0,1)
23: LIT(-2)
24: ADD                ; N-2 auf Datenkeller
25: CALL(5,1,1)        ; Rekursiver Aufruf f(N-2) mit Parameter auf Da
26: ADD                ; Ergebnisse der Aufrufe addieren
27: RET                ; Ruecksprung
```

Aufgabe 36

```
import Trace

infixr 0 !


type State = (Int, [Int], [Int])
data Code  = ADD | SUB | DIV | MUL | NOT | AND | OR | LTH | GTH | LEQ | GEQ
           | EQU | NEQ | JMP Int | JFALSE Int | CALL Int Int Int | RET
           | LOAD Int Int | STORE Int Int | LIT Int
             deriving Show

type Prog = [Code]
type ISemantik = Prog -> State -> State
type CSemantik = Code -> State -> State
type MSemantik = Prog -> [Int] -> [Int]

base :: [Int] -> Int -> Int
base p 0 = 1
base p (dif+1) = (base p dif) + (p ! (base p dif))

csem :: CSemantik
csem ADD (m, (z2:(z1:d)), p) = (m+1,(z1+z2):d, p)
csem SUB (m, (z2:(z1:d)), p) = (m+1,(z1-z2):d, p)
csem MUL (m, (z2:(z1:d)), p) = (m+1,(z1*z2):d, p)
csem DIV (m, (z2:(z1:d)), p) = (m+1,(mod z1 z2):d, p)

csem LTH (m, (z2:(z1:d)), p) = (m+1,(fromBool (z1 <  z2)):d, p)
csem GTH (m, (z2:(z1:d)), p) = (m+1,(fromBool (z1 >  z2)):d, p)
csem LEQ (m, (z2:(z1:d)), p) = (m+1,(fromBool (z1 <= z2)):d, p)
csem GEQ (m, (z2:(z1:d)), p) = (m+1,(fromBool (z1 >= z2)):d, p)
csem EQU (m, (z2:(z1:d)), p) = (m+1,(fromBool (z1 == z2)):d, p)
csem NEQ (m, (z2:(z1:d)), p) = (m+1,(fromBool (z1 /= z2)):d, p)

csem AND (m, (0:(0:d)), p) = (m+1, 0:d, p)
csem AND (m, (0:(1:d)), p) = (m+1, 0:d, p)
csem AND (m, (1:(0:d)), p) = (m+1, 0:d, p)
csem AND (m, (1:(1:d)), p) = (m+1, 1:d, p)
csem OR  (m, (0:(0:d)), p) = (m+1, 0:d, p)
csem OR  (m, (0:(1:d)), p) = (m+1, 1:d, p)
csem OR  (m, (1:(0:d)), p) = (m+1, 1:d, p)
csem OR  (m, (1:(1:d)), p) = (m+1, 1:d, p)
csem NOT (m, (0:d), p) = (m+1, 1:d, p)
csem NOT (m, (1:d), p) = (m+1, 0:d, p)

csem (JFALSE n) (m, (0:d), p) = (n,   d, p)
csem (JFALSE n) (m, (1:d), p) = (m+1, d, p)
csem (JMP n) (m, d, p) = (n, d, p)

csem (CALL ca dif loc) (m, d, p) = (ca, d, ((base p dif) + loc + 2 : loc+2 : m+1 : (repl
```

```
csem RET (m,d,p@(p1:p2:p3:_)) = (p3, d, drop (1+p2) p)

csem (LOAD  dif off) (m,d,p) = (m+1, (p ! ((base p dif) + 2 + off)) : d, p)
csem (STORE dif off) (m,z:d,p) = (m+1, d, set p z ((base p dif) + 2 + off))

csem (LIT z) (m,d,p) = (m+1, z:d, p)

isem :: ISemantik
-- Einzelschrittsemantik
isem prog (m,d,p) =
    if  m == 0 || m > length prog
        then (m,d,p)
        else isem prog ((csem (prog ! m) (tns (m,d,p))))


startState :: [Int] -> State
startState x = (1,[],[0,0,0]++x)

msem :: MSemantik
msem p s = let (_, _, _:_:_:res) = isem p (startState s)
             in res

-- Hilfsfunktionen
-- nimm das i-te Element einer Liste, gezaehlt wird ab 1
(!) :: [a] -> Int -> a
l ! i = l !! (i-1)

-- setze das i-te Element einer Liste auf x
set :: [a] -> a -> Int -> [a]
set (_:l) x 1 = x:l
set (y:l) x (n+2) = y:(set l x (n+1))

fromBool :: Bool -> Int
fromBool = fromEnum

tns :: (Show a) => a -> a
tns a = trace ((show a)++"\n") a
```

Lösungsvorschläge zur 13. Übung „Compilerbau“, SS 2004

Aufgabe 37

a)

- Erweiterung von $\mathfrak{E}_a : AExpr \times U \times S \dashrightarrow \mathbb{Z}$ um:

$$\mathfrak{E}_a[\![E_1 \mod E_2]\!]\rho\sigma := \mathfrak{E}_a[\![E_1]\!]\sigma\rho \mod [\![E_2]\!]\sigma\rho$$

- Definiere die Semantik für den Maschinenbefehl *MOD*:

$$[\![MOD]\!](m, d : a : b, p) := (m+1, d : (a \mod b), p)$$

- Erweitere die Übersetzungs-Hilfsfunktion durch:

$$\begin{aligned} \underline{et}(E_1 \mod E_2, st, a, l) := & \underline{et}(E_1, st, a, l) \\ & \underline{et}(E_2, st, a', l) \\ & a'' : \mathbf{MOD} \end{aligned}$$

b) Das Programm in der Aufgabenstellung sei beginnend mit 1 zeilenweise durchnummeriert. Dann setze:
$B := \Delta\Gamma,\ \Delta := Z2 - Z12,\ \Delta_p := Z3 - Z12,\ \Gamma := Z13 - Z17,\ B' := \Delta'\Gamma',\ \Delta' := Z4,\ \Gamma' := Z5 - Z12,\ \Gamma_{if} := Z6 - Z12$

$$\underline{trans}(P) = \underline{trans}(\mathbf{in/out}\ X, Y; B)$$

$$\begin{aligned} = & 1 : \mathbf{CALL}(a_\Gamma, 0, 1) \\ & 2 : \mathbf{JMP}(0) \\ & \underline{bt}(B, st_{I/O}, a_\Gamma, 1) \end{aligned}$$

$$\begin{aligned} = & 1 : \mathbf{CALL}(a_\Gamma, 0, 1) \\ & 2 : \mathbf{JMP}(0) \\ & \underline{dt}(\Delta, \underline{up}(\mathbf{var}\ Z; \mathbf{proc}\ G; B', st_{I/O}, a_1, 1), a_1, 1) \\ & \underline{ct}(\Gamma, \underline{up}(\mathbf{var}\ Z; \mathbf{proc}\ G; B', st_{I/O}, a_1, 1), a_\Gamma, 1) \\ & a' : \mathbf{RET} \end{aligned}$$

$$\begin{aligned} = & 1 : \mathbf{CALL}(a_\Gamma, 0, 1) \\ & 2 : \mathbf{JMP}(0) \\ & \underline{dt}(\Delta, \underline{up}(\mathbf{proc}\ G; B';, \underline{up}(\mathbf{var}\ Z;, st_{I/O}, a_1, 1), a_1, 1), a_1, 1) \\ & \underline{ct}(\Gamma, \underline{up}(\mathbf{proc}\ G; B';, \underline{up}(\mathbf{var}\ Z;, st_{I/O}, a_1, 1), a_1, 1) a_\Gamma, 1) \\ & a' : \mathbf{RET} \end{aligned}$$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$\quad 2 : \mathbf{JMP}(0)$
$\quad \underline{dt}(\Delta, \underline{up}(\mathbf{proc}\ G; B'; , \underbrace{st_{I/O}[Z/(\underline{var}, 1, 1)]}_{=:st_1}, a_1, 1), a_1, 1)$
$\quad \underline{ct}(\Gamma, \underline{up}(\mathbf{proc}\ G; B'; , st_1, a_1, 1), a_\Gamma, 1)$
$\quad a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$\quad 2 : \mathbf{JMP}(0)$
$\quad \underline{dt}(\Delta_p, \underbrace{st_1[G/(\underline{proc}, a_1, 1, 1)]}_{=:st_2}, a_1, 1)$
$\quad \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$\quad a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$\quad 2 : \mathbf{JMP}(0)$
$\quad \underline{bt}(B', st_2, a_1, 1)$
$\quad \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$\quad a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$\quad 2 : \mathbf{JMP}(0)$
$\quad \underline{dt}(\Delta', \underline{up}(\Delta', st_2, a_2, 1), a_2, 2)$
$\quad \underline{ct}(\Gamma', \underline{up}(\Delta', st_2, a_2, 1), a_1, 2)$
$\quad \widetilde{a} : \mathbf{RET}$
$\quad \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$\quad a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$\quad 2 : \mathbf{JMP}(0)$
$\quad \underline{dt}(\Delta', \underbrace{st_2[Z/(\underline{var}, 2, 1)]}_{=:st_3}, a_2, 2) = \varepsilon$
$\quad \underline{ct}(\Gamma', st_3, a_1, 2)$
$\quad \widetilde{a} : \mathbf{RET}$
$\quad \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$\quad a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$\underline{ct}(Z := 1;, st_3, a_1, 2)$
$\underline{ct}(\Gamma_{if}, st_3, a'', 2)$
$\widetilde{a} : \mathbf{RET}$
$\underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$\underline{et}(1, st_3, a_1, 2)$
$a''' : \mathbf{STORE}(0, 1)$
$\underline{ct}(\Gamma_{if}, st_3, a'', 2)$
$\widetilde{a} : \mathbf{RET}$
$\underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$a_1 : \mathbf{LIT}(1)$
$a''' : \mathbf{STORE}(0, 1)$
$\underline{ct}(\Gamma_{if}, st_3, a'', 2)$
$\widetilde{a} : \mathbf{RET}$
$\underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$a_1 : \mathbf{LIT}(1)$
$a^{(3)} : \mathbf{STORE}(0, 1)$
$\underline{sbt}(Y = 0, st_3, a'', 2)$
$a^{(4)} : \mathbf{JFALSE}(a^{(5)})$
$\underline{ct}(Y := X;, st_3, a^{(4)} + 1, 2)$
$a^{(5)} - 1 : \mathbf{JMP}(a^{(6)})$
$\underline{ct}(Z := X \mod Y; X := Y; Y := Z; G();, st_3, a^{(5)}, 2)$
$\widetilde{a} = a^{(6)} : \mathbf{RET}$
$\underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$a_1 : \mathbf{LIT}(1)$
$a^{(3)} : \mathbf{STORE}(0,1)$
$\underline{et}(Y, st_3, a'', 2)$
$\underline{et}(0, st_3, a^{(7)}, 2)$
$a^{(8)} : \mathbf{EQ}$;Testen auf Gleichheit (durch LT simulierbar)
$a^{(4)} : \mathbf{JFALSE}(a^{(5)})$
$\underline{ct}(Y := X;, st_3, a^{(4)} + 1, 2)$
$a^{(5)} - 1 : \mathbf{JMP}(a^{(6)})$
$\underline{ct}(Z := X \mod Y; X := Y; Y := Z; G();, st_3, a^{(5)}, 2)$
$\widetilde{a} = a^{(6)} : \mathbf{RET}$
$a^{(6)} : \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$a_1 : \mathbf{LIT}(1)$
$a^{(3)} : \mathbf{STORE}(0,1)$
$a'' : \mathbf{LOAD}(2,2)$
$a^{(7)} : \mathbf{LIT}(0)$
$a^{(8)} : \mathbf{EQ}$
$a^{(4)} : \mathbf{JFALSE}(a^{(5)})$
$\underline{et}(X, st_3, a^{(4)} + 1, 2)$
$a^{(9)} : \mathbf{STORE}(2,2)$
$a^{(5)} - 1 : \mathbf{JMP}(a^{(6)})$
$\underline{ct}(Z := X \mod Y; X := Y; Y := Z; G();, st_3, a^{(5)}, 2)$
$\widetilde{a} = a^{(6)} : \mathbf{RET}$
$a^{(6)} : \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$a_1 : \mathbf{LIT}(1)$
$a^{(3)} : \mathbf{STORE}(0,1)$
$a'' : \mathbf{LOAD}(2,2)$
$a^{(7)} : \mathbf{LIT}(0)$
$a^{(8)} : \mathbf{EQ}$
$a^{(4)} : \mathbf{JFALSE}(a^{(5)})$
$a^{(4)}+1 : \mathbf{LOAD}(2,1)$
$a^{(9)} : \mathbf{STORE}(2,2)$
$a^{(5)}-1 : \mathbf{JMP}(a^{(6)})$
$\underline{ct}(Z := X \mod Y;, st_3, a^{(5)}, 2)$
$\underline{ct}(X := Y;, st_3, a^{(10)}, 2)$
$\underline{ct}(Y := Z;, st_3, a^{(11)}, 2)$
$\underline{ct}(G();, st_3, a^{(12)}, 2)$
$\widetilde{a} = a^{(6)} : \mathbf{RET}$
$a^{(6)} : \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$=1 : \mathbf{CALL}(a_\Gamma, 0, 1)$
$2 : \mathbf{JMP}(0)$
$a_1 : \mathbf{LIT}(1)$
$a^{(3)} : \mathbf{STORE}(0,1)$
$a'' : \mathbf{LOAD}(2,2)$
$a^{(7)} : \mathbf{LIT}(0)$
$a^{(8)} : \mathbf{EQ}$
$a^{(4)} : \mathbf{JFALSE}(a^{(5)})$
$a^{(4)}+1 : \mathbf{LOAD}(2,1)$
$a^{(9)} : \mathbf{STORE}(2,2)$
$a^{(5)}-1 : \mathbf{JMP}(a^{(6)})$
$\underline{et}(X \mod Y, st_3, a^{(5)}, 2)$
$a^{(13)} : \mathbf{STORE}(0,1)$
$\underline{et}(Y, st_3, a^{(10)}, 2)$
$a^{(14)} : \mathbf{STORE}(2,1)$
$\underline{et}(Z, st_3, a^{(11)}, 2)$
$a^{(15)} : \mathbf{STORE}(2,2)$
$a^{(12)} : \mathbf{CALL}(a_1, 1, 1)$
$\widetilde{a} = a^{(6)} : \mathbf{RET}$
$a^{(6)} : \underline{ct}(\Gamma, st_2, a_\Gamma, 1)$
$a' : \mathbf{RET}$

$$
\begin{aligned}
=1 &: \mathbf{CALL}(a_\Gamma, 0, 1) \\
2 &: \mathbf{JMP}(0) \\
a_1 &: \mathbf{LIT}(1) \\
a^{(3)} &: \mathbf{STORE}(0,1) \\
a'' &: \mathbf{LOAD}(2,2) \\
a^{(7)} &: \mathbf{LIT}(0) \\
a^{(8)} &: \mathbf{EQ} \\
a^{(4)} &: \mathbf{JFALSE}(a^{(5)}) \\
a^{(4)}+1 &: \mathbf{LOAD}(2,1) \\
a^{(9)} &: \mathbf{STORE}(2,2) \\
a^{(5)}-1 &: \mathbf{JMP}(a^{(6)}) \\
\underline{et}(X, st_3, a^{(5)}, 2) & \\
\underline{et}(Y, st_3, a^{(16)}, 2) & \\
a^{(17)} &: \mathbf{MOD} \\
a^{(13)} &: \mathbf{STORE}(0,1) \\
a^{(10)} &: \mathbf{LOAD}(2,2) \\
a^{(14)} &: \mathbf{STORE}(2,1) \\
a^{(11)} &: \mathbf{LOAD}(0,1) \\
a^{(15)} &: \mathbf{STORE}(2,2) \\
a^{(12)} &: \mathbf{CALL}(a_1, 1, 1) \\
\widetilde{a} = a^{(6)} &: \mathbf{RET} \\
a^{(6)} &: \underline{ct}(\Gamma, st_2, a_\Gamma, 1) \\
a' &: \mathbf{RET}
\end{aligned}
$$

$$
\begin{aligned}
=1 &: \mathbf{CALL}(a_\Gamma, 0, 1)\\
2 &: \mathbf{JMP}(0)\\
a_1 &: \mathbf{LIT}(1)\\
a^{(3)} &: \mathbf{STORE}(0,1)\\
a'' &: \mathbf{LOAD}(2,2)\\
a^{(7)} &: \mathbf{LIT}(0)\\
a^{(8)} &: \mathbf{EQ}\\
a^{(4)} &: \mathbf{JFALSE}(a^{(5)})\\
a^{(4)}+1 &: \mathbf{LOAD}(2,1)\\
a^{(9)} &: \mathbf{STORE}(2,2)\\
a^{(5)}-1 &: \mathbf{JMP}(a^{(6)})\\
a^{(5)} &: \mathbf{LOAD}(2,1)\\
a^{(16)} &: \mathbf{LOAD}(2,2)\\
a^{(17)} &: \mathbf{MOD}\\
a^{(13)} &: \mathbf{STORE}(0,1)\\
a^{(10)} &: \mathbf{LOAD}(2,2)\\
a^{(14)} &: \mathbf{STORE}(2,1)\\
a^{(11)} &: \mathbf{LOAD}(0,1)\\
a^{(15)} &: \mathbf{STORE}(2,2)\\
a^{(12)} &: \mathbf{CALL}(a_1, 1, 1)\\
\widetilde{a} = a^{(6)} &: \mathbf{RET}\\
\underline{ct}(Z := 0;, st_2, a_\Gamma, 1)\\
\underline{ct}(G();, st_2, a^{(18)}, 1)\\
\underline{ct}(X := Z;, st_2, a^{(19)}, 1)\\
a' &: \mathbf{RET}
\end{aligned}
$$

$$
\begin{aligned}
=1 &: \mathbf{CALL}(a_\Gamma, 0, 1) \\
2 &: \mathbf{JMP}(0) \\
a_1 &: \mathbf{LIT}(1) \\
a^{(3)} &: \mathbf{STORE}(0, 1) \\
a'' &: \mathbf{LOAD}(2, 2) \\
a^{(7)} &: \mathbf{LIT}(0) \\
a^{(8)} &: \mathbf{EQ} \\
a^{(4)} &: \mathbf{JFALSE}(a^{(5)}) \\
a^{(4)} + 1 &: \mathbf{LOAD}(2, 1) \\
a^{(9)} &: \mathbf{STORE}(2, 2) \\
a^{(5)} - 1 &: \mathbf{JMP}(a^{(6)}) \\
a^{(5)} &: \mathbf{LOAD}(2, 1) \\
a^{(16)} &: \mathbf{LOAD}(2, 2) \\
a^{(17)} &: \mathbf{MOD} \\
a^{(13)} &: \mathbf{STORE}(0, 1) \\
a^{(10)} &: \mathbf{LOAD}(2, 2) \\
a^{(14)} &: \mathbf{STORE}(2, 1) \\
a^{(11)} &: \mathbf{LOAD}(0, 1) \\
a^{(15)} &: \mathbf{STORE}(2, 2) \\
a^{(12)} &: \mathbf{CALL}(a_1, 1, 1) \\
\widetilde{a} = a^{(6)} &: \mathbf{RET} \\
a_\Gamma &: \mathbf{LIT}(0) \\
a^{(20)} &: \mathbf{STORE}(0, 1) \\
a^{(18)} &: \mathbf{CALL}(a_1, 0, 1) \\
a^{(19)} &: \mathbf{LOAD}(0, 1) \\
a^{(21)} &: \mathbf{STORE}(1, 1) \\
a' &: \mathbf{RET}
\end{aligned}
$$

$$
\begin{aligned}
=01 &: \textbf{CALL}(22,0,1)\\
02 &: \textbf{JMP}(0)\\
03 &: \textbf{LIT}(1)\\
04 &: \textbf{STORE}(0,1)\\
05 &: \textbf{LOAD}(2,2)\\
06 &: \textbf{LIT}(0)\\
07 &: \textbf{EQ}\\
08 &: \textbf{JFALSE}(12)\\
09 &: \textbf{LOAD}(2,1)\\
10 &: \textbf{STORE}(2,2)\\
11 &: \textbf{JMP}(21)\\
12 &: \textbf{LOAD}(2,1)\\
13 &: \textbf{LOAD}(2,2)\\
14 &: \textbf{MOD}\\
15 &: \textbf{STORE}(0,1)\\
16 &: \textbf{LOAD}(2,2)\\
17 &: \textbf{STORE}(2,1)\\
18 &: \textbf{LOAD}(0,1)\\
19 &: \textbf{STORE}(2,2)\\
20 &: \textbf{CALL}(3,1,1)\\
21 &: \textbf{RET}\\
22 &: \textbf{LIT}(0)\\
23 &: \textbf{STORE}(0,1)\\
24 &: \textbf{CALL}(3,0,1)\\
25 &: \textbf{LOAD}(0,1)\\
26 &: \textbf{STORE}(1,1)\\
27 &: \textbf{RET}
\end{aligned}
$$

c)
```
Eingabewert f{\"u}r X: 4
Eingabewert f{\"u}r Y: 2

 Datenkeller:
 Prozedurkeller: 0 0 0 4 2
Codezeile 001   CALL(22,0,1)
 Datenkeller:
 Prozedurkeller: 4 3 2 0 0 0 0 4 2
Codezeile 022   LIT(0)
 Datenkeller: 0
 Prozedurkeller: 4 3 2 0 0 0 0 4 2
Codezeile 023   STORE(0,1)
 Datenkeller:
 Prozedurkeller: 4 3 2 0 0 0 0 4 2
Codezeile 024   CALL(3,0,1)
 Datenkeller:
```

```
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 4 2
Codezeile 003   LIT(1)
 Datenkeller: 1
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 4 2
Codezeile 004   STORE(0,1)
 Datenkeller:
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 005   LOAD(2,2)
 Datenkeller: 2
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 006   LIT(0)
 Datenkeller: 2 0
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 007   EQ
 Datenkeller: 0
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 008   JFALSE(12)
 Datenkeller:
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 012   LOAD(2,1)
 Datenkeller: 4
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 013   LOAD(2,2)
 Datenkeller: 4 2
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 014   MOD
 Datenkeller: 0
 Prozedurkeller: 4 3 25 1 4 3 2 0 0 0 0 4 2
Codezeile 015   STORE(0,1)
 Datenkeller:
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 4 2
Codezeile 016   LOAD(2,2)
 Datenkeller: 2
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 4 2
Codezeile 017   STORE(2,1)
 Datenkeller:
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 2 2
Codezeile 018   LOAD(0,1)
 Datenkeller: 0
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 2 2
Codezeile 019   STORE(2,2)
 Datenkeller:
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 020   CALL(3,1,1)
 Datenkeller:
 Prozedurkeller: 8 3 21 0 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 003   LIT(1)
 Datenkeller: 1
 Prozedurkeller: 8 3 21 0 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 004   STORE(0,1)
 Datenkeller:
```

```
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 005   LOAD(2,2)
 Datenkeller: 0
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 006   LIT(0)
 Datenkeller: 0 0
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 007   EQ
 Datenkeller: 1
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 008   JFALSE(12)
 Datenkeller:
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 009   LOAD(2,1)
 Datenkeller: 2
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 0
Codezeile 010   STORE(2,2)
 Datenkeller:
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 2
Codezeile 011   JMP(21)
 Datenkeller:
 Prozedurkeller: 8 3 21 1 4 3 25 0 4 3 2 0 0 0 0 2 2
Codezeile 021   RET
 Datenkeller:
 Prozedurkeller: 4 3 25 0 4 3 2 0 0 0 0 2 2
Codezeile 021   RET
 Datenkeller:
 Prozedurkeller: 4 3 2 0 0 0 0 2 2
Codezeile 025   LOAD(0,1)
 Datenkeller: 0
 Prozedurkeller: 4 3 2 0 0 0 0 2 2
Codezeile 026   STORE(1,1)
 Datenkeller:
 Prozedurkeller: 4 3 2 0 0 0 0 0 2
Codezeile 027   RET Datenkeller:
 Prozedurkeller: 0 0 0 0 2
Codezeile 002   JMP(0)
 Datenkeller:
 Prozedurkeller: 0 0 0 0 2

Ausgabewert fuer X: 0
Ausgebewert fuer Y: 2
```

Es gilt also $\mathfrak{I}[\![\underline{trans}(P)]\!](1,\varepsilon,0:0:0:4:2) = (0,\varepsilon,0:0:0:0:2)$. Das Programm berechnet den größten gemeinsamen Teiler von zwei Zahlen (Euklidischer Algorithmus).