RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

1. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 28.04.2004, vor der Frontalübung

Aufgabe 1 (2+2 Punkte)

a) Geben Sie eine kompakte, aber nachvollziehbare Definition an, die die Gleitpunktkonstanten einer Scheme-ähnlichen Sprache mit einem regulären Ausdruck α beschreibt:

> Eine Gleitpunktkonstante besteht aus einem optionalen Vorzeichen (`+` oder `-`), dem Dezimalteil, und einem Suffix, das die Genauigkeit beschreibt.
>
> Der Dezimalteil besteht *entweder* aus einer Ziffernsequenz (eine nicht-leeren Sequenz der Zeichen `0` bis `9`), *oder* einem Dezimalpunkt `.`, gefolgt von einer Ziffernsequenz, *oder* einer Ziffernsequenz, einem Dezimalpunkt, und einer Ziffernsequenz (dem Dezimalbruch), *oder* einer Ziffernsequenz, gefolgt von einem Dezimalpunkt.
>
> Das Suffix besteht aus einem der Buchstaben `e`, `s`, `f`, `d` oder `l`, gefolgt von einem optionalen Vorzeichen und einer Ziffernsequenz. Weiterhin gilt, daß das Suffix optional ist, falls im Dezimalteil ein Dezimalpunkt auftritt.

b) Geben Sie Beispiele $w_i \in [\![\alpha]\!], v_i \notin [\![\alpha]\!], i \in \{1,2,3\}$, mit denen man verschiedene optionale und Auswahlkomponenten von α testen kann. Dokumentieren Sie außerdem jeweils, welche Komponente von α getestet wird.

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

Die regulären Ausdrücke RegE(Σ) sollen neben den Operatoren α^*, $\alpha \vee \beta$, $\alpha \cdot \beta$ um den Komplementoperator $\neg\alpha$ erweitert werden. Für $\alpha \in \text{RegE}(\Sigma)$ gelte $[\![\neg\alpha]\!] := \Sigma^* \setminus [\![\alpha]\!]$.

Sei nun $\Sigma = \{a, \ldots, z\}, \beta_0 = foo^*$ und $\beta = \neg\beta_0 d^*$.

a) Konstruieren Sie einen NFA $\mathfrak{A}(\beta_0)$.

b) Konstruieren Sie den Komplementautomaten $\mathfrak{A}(\neg\beta_0)$ mit $L(\mathfrak{A}(\neg\beta_0)) = [\![\neg\beta_0]\!]$ so, daß er als Baustein in der Thompson-Konstruktion geeignet ist.

c) Benutzen Sie das Ergebnis aus Teil (b) für die Konstruktion des NFA $\mathfrak{A}(\beta)$ nach der Thompson-Methode.

d) Erkennt der Automat $\mathfrak{A}(\beta)$ das Wort $w = fooled$? Begründen Sie Ihre Antwort.

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

Ein erweitertes Matching-Problem sei gegeben durch die folgenden regulären Ausdrücke:

$$\alpha_1 = a \qquad \alpha_2 = a^* c \qquad \alpha_3 = bc$$

a) Konstruieren Sie den zugehörigen Produktautomaten $\mathfrak{A} \in \mathrm{DFA}(\{a,b,c\})$ mit $L(\mathfrak{A}) = \bigcup_{i=1}^{3} [\![\alpha_i]\!]$.

b) Berechnen Sie die Menge P der produktiven Zustände für $\mathfrak{A}$.

c) Geben Sie einen Lauf des aus $\mathfrak{A}$ gewonnenen Backtrack-Automaten $\mathfrak{B}$ an, der die flm-Analyse $v = T_{i_1} \ldots T_{i_k}$ für das Wort $w = aaabc$ berechnet.

d) Zeigen Sie, daß $\mathfrak{B}$ für die Analyse eines Wortes $w_n = a^n$ höchstens $O(n^2)$ Schritte benötigt.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

2. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 05.05.2004, vor der Frontalübung

Aufgabe 4 (2+2 Punkte)

a) Beweisen oder widerlegen Sie:

> Seien $\alpha_1, \ldots, \alpha_n \in \mathrm{RegE}(\Sigma)$ reguläre Ausdrücke und $w \in \Sigma^\star$ ein Wort. Wenn es eine Zerlegung von w bezüglich $(\alpha_1, \ldots, \alpha_n)$ gibt, so gibt es auch eine lm–Zerlegung von w bezüglich $(\alpha_1, \ldots, \alpha_n)$.

b) In der Vorlesung wurde auf eine Arbeit hingewiesen, in welcher der *worst case*-Zeitaufwand der flm-Analyse von $O(n^2)$ auf $O(n)$ reduziert wird. Erörtern Sie den Nutzen des Einsatzes dieser Optimierung in Lexerprogrammen für aktuelle Programmiersprachen.

Aufgabe 5 (2+3 Punkte)

In der Vorlesung wurde für die flm-Analyse ein Backtrackautomat aus einem Produkt-DFA konstruiert. Da die Thompson-Konstruktion jedoch NFAs erzeugt, soll nun ein Backtrack-Automat *direkt* aus NFAs konstruiert werden, ohne Verwendung der Potenzmengenkonstruktion:

Gegeben seien die durch Thompson-Konstruktion erzeugten Automaten $\mathfrak{A}(\alpha_1), \ldots, \mathfrak{A}(\alpha_n)$ mit

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

a) Konstruieren Sie zunächst den Vereinigungsautomaten

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

der die Sprache $L(\mathfrak{A}) = \bigcup_{i=1}^{n} L(\mathfrak{A}(\alpha_i))$ erkennt.

b) Konstruieren Sie darauf aufbauend einen Backtrackautomaten $\mathfrak{B} := \langle Conf, \vdash \rangle$ mit $Conf$ als Menge der Konfigurationen, der Übergangsrelation $\vdash \subseteq Conf^2$ und der Anfangskonfiguration $init_w$ für eine Eingabe $w \in \Sigma^*$, der als Ausgabe eine flm-Analyse $v = T_{i_1} \ldots T_{i_m} \in \Delta^*$ liefert, oder einen Fehler *lexerr*, falls eine solche nicht existiert.

Aufgabe 6 (5 Punkte)

Schreiben Sie für die *Universal Toy Scheme Language* ein `flex`-Programm `utsl.l`, welches vollständig geklammerte Ausdrücke in Präfixform von der Standardeingabe einliest. Die `main`-Funktion soll jeweils die gescannten Symbole als Liste von Paaren $(token \;.\; attr)$ ausgeben. Falls ein Symbol keinen sinnvollen Attributwert besitzt, soll stattdessen _ ausgegeben werden.

Verwenden Sie das folgende Token-Alphabet:

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

Die Analyse der Eingabe soll anhand folgender regulärer Ausdrücke geschehen (Σ: Extended ASCII-Alphabet):

$$
\begin{aligned}
\underline{\textit{tokenspace}} &:= \underline{\textit{comment}} \mid \underline{\textit{whitespace}}^{+} \\
\underline{\textit{comment}} &:= \{;\}\Sigma^{*}\{\textit{end-of-line}\} \\
\underline{\textit{lparen}} &:= \{(\} \\
\underline{\textit{rparen}} &:= \{)\} \\
\underline{\textit{boolean}} &:= \{\texttt{\#f},\texttt{\#t}\} \\
\underline{\textit{integer}} &:= \underline{\textit{sign}}?\underline{\textit{digit}}^{+} \\
\underline{\textit{float}} &:= \underline{\textit{sign}}?(\underline{\textit{digit}}^{+}\{.\}\underline{\textit{digit}}^{*} \mid \underline{\textit{digit}}^{*}\{.\}\underline{\textit{digit}}^{+})(\{\texttt{e},\texttt{E}\}\underline{\textit{sign}}?\underline{\textit{digit}}^{+})? \\
\underline{\textit{sign}} &:= \{-,+\} \\
\underline{\textit{string_element}} &:= (\Sigma \setminus \{\backslash,\texttt{"}\}) \mid \{\backslash\texttt{"}\} \mid \{\backslash\backslash\} \\
\underline{\textit{string}} &:= \{\texttt{"}\}\ \underline{\textit{string_element}}^{*}\ \{\texttt{"}\} \\
\underline{\textit{initial}} &:= \underline{\textit{alpha}} \mid \underline{\textit{special_initial}} \\
\underline{\textit{special_initial}} &:= \{-,!,\$,\%,\&,*,/,:,<,=,>,?,\hat{},_\} \\
\underline{\textit{subsequent}} &:= \underline{\textit{initial}} \mid \underline{\textit{digit}} \mid \{-,+,.,@\} \\
\underline{\textit{peculiar_identifier}} &:= \{-,+\} \\
\underline{\textit{identifier}} &:= \underline{\textit{initial}}\ \underline{\textit{subsequent}}^{*} \mid \underline{\textit{peculiar_identifier}}
\end{aligned}
$$

Hinweise:

- Kommentare und *whitespace* dienen als Token-Begrenzer und werden wie üblich ignoriert.
- Der Quelltext sollte sich mit `flex -outsl-lexer.c utsl.l && cc utsl-lexer.c -lfl -o utsl-lexer` in ein ausführbares Programm übersetzen lassen.
- Sie können die C-Funktionen `int atoi (const char *)` und `double atof (const char *)` zur Konvertierung von Integer- bzw. Floatwerten verwenden.

Beispiellauf:

```
$ ./utsl-lexer
(/ (- 128 2)
; comment
   3.0)
^D
((LPAREN . _) (IDENTIFIER . "/") (LPAREN . _) (IDENTIFIER . "-") (INTEGER . 128)
 (INTEGER . 2) (RPAREN . _) (FLOAT . 3.000000) (RPAREN . _))
```

Jede Gruppe schickt den Quelltext und *je 3 verschiedene Beispielläufe* als Attachments zusätzlich an `rieger+cb2004@i2.informatik.rwth-aachen.de`. Im `Subject` der Mail sollen die Aufgabennummer und die Matrikelnummern der Autoren aufgeführt sein (ebenso im Quelltext), also z. B.

```
Subject: A6: 987654 876543 765432
```

Unter `http://www-i2.informatik.rwth-aachen.de/Teaching/Course/CB/2004/utsl/` finden Sie ein Programmskelett, das sie verwenden sollten.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

3. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 12.05.2004, vor der Frontalübung

Aufgabe 7 (4 Punkte)

In der Vorlesung wurde die Korrektheit des NTA(G) bereits gezeigt. Beweisen Sie nun dessen Vollständigkeit, d.h.

$$S \underset{l}{\overset{z}{\Rightarrow}} w \quad \rightsquigarrow \quad (w, S, \varepsilon) \overset{*}{\vdash} (\varepsilon, \varepsilon, z)$$

Aufgabe 8 (2+1+2 Punkte)

Gegeben ist die Grammatik G:

$$\begin{aligned} S &\to B \mid S\mathtt{a}A \\ A &\to \mathtt{d}ACB \mid \varepsilon \\ B &\to \mathtt{b}C \mid C \mid AC\mathtt{a} \\ C &\to BC \mid \mathtt{a} \mid D\mathtt{en} \mid \varepsilon \\ D &\to \mathtt{c}S\mathtt{h} \end{aligned}$$

a) Konstruieren Sie den NTA(G) und geben Sie die erfolgreiche Berechnung einer Analyse z für das Wort $w = \mathtt{badaachen} \in \Sigma^*$ an.

b) Geben Sie den Ableitungsbaum zu der in Teil (a) berechneten Analyse z an.

c) Skizzieren Sie einen allgemeinen Algorithmus, der zu einer gegebenen l-Analyse z' für ein Wort $v \in \Sigma^*$ eine äquivalente[1] r-Analyse z'' ausgibt, so daß $S \underset{r}{\overset{z''}{\Rightarrow}} v$. Wenden Sie anschließend diesen Algorithmus auf z an.

Aufgabe 9 (1+1+2 Punkte)

Beweisen Sie die folgenden Eigenschaften von $\underline{\text{first}}$:

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

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

c) $v \in \underline{\text{first}}_k(\alpha) \quad \rightsquigarrow \quad \exists x \in \Sigma^* : \alpha \overset{*}{\Rightarrow} x, \{v\} = \underline{\text{first}}_k(x)$

[1] zwei Analysen heißen *äquivalent*, falls sie den gleichen Ableitungsbaum besitzen

Aufgabe 10 (4 Punkte)

Um aussagekräftige Fehlermeldungen zu erhalten, ist es notwendig, die vom Lexer erzeugten Token mit präzisen Positionsinformationen zu versehen.

Erweitern Sie den Lexer aus Aufgabe 6 derart, daß für jedes Token die folgenden *location*-Informationen zur Verfügung stehen:

```
struct {
  int first_line, first_column,
      last_line,  last_column;
} yyltype;

yyltype yylloc;
```

Dabei soll für zwei direkt aufeinanderfolgende (d. h. nicht durch *whitespace* separierte) Token t_i und t_{i+1} gelten, daß

$$\texttt{yylloc}_{t_{i+1}}\texttt{.first_column} = \texttt{yylloc}_{t_i}\texttt{.last_column} + 1$$

Geben Sie diese Informationen als zusätzliches Attribut für jedes Token aus:

```
$ ./utsl-lexer
(/ (- 128 2)
; comment
 3.0)
((LPAREN (1 1 1 1) _)(IDENTIFIER (1 2 1 2) /)(LPAREN (1 4 1 4) _)
 (IDENTIFIER (1 5 1 5) -)(INTEGER (1 7 1 9) 128)(INTEGER (1 11 1 11) 2)
 (RPAREN (1 12 1 12) _)(FLOAT (3 2 3 4) 3.000000)(RPAREN (3 5 3 5) _))
;; # of lines = 3
```

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

4. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 19.05.2004, vor der Frontalübung

Aufgabe 11 (2+2 Punkte)

a) Geben Sie eine zur Definition äquivalente Charakterisierung der Menge aller reduzierten LL(0)–Grammatiken sowie der Menge aller LL(0)– (d.h. der durch LL(0)–Grammatiken erzeugbaren) Sprachen an.

b) Zeigen Sie, daß jede reguläre Sprache durch eine LL(1)–Grammatik erzeugt wird.

Aufgabe 12 (2+2+2+2+2 Punkte)

Die Syntax einer C-ähnlichen imperativen Programmiersprache P_0 ist durch die Syntaxdiagramme in Abbildung 1 gegeben.

a) Geben Sie die Syntax von P_0 durch eine kontextfreie Grammatik G an.

b) Berechnen Sie zu jedem Nichtterminalsymbol A aus G die Mengen $\mathbf{fi}(A)$ und $\mathbf{fo}(A)$.

c) Prüfen Sie, ob es sich bei Ihrer Grammatik um eine LL(1)–Grammatik handelt.

d) Geben Sie eine LL(1)–Grammatik G^1 für P_0 an.

e) Konstruieren Sie den zugehörigen deterministischen Top–down–Analyseautomaten DTA(G^1).

Aufgabe 13 (2 Punkte)

Formen Sie die in Abbildung 1 gegebene Grammatik G wie folgt um:

- *statement*s, *condition*s und *expression*s werden vollständig geklammert, d. h. jede rechte Seite der Produktionen für *statement*, *condition* und *expression* beginnt mit einer öffnenden Klammer `(` und endet mit einer schließenden Klammer `)`.
- Alle Operatoren werden zu Präfixoperatoren. Verwenden Sie `set!` anstelle von `:=`.
- Eliminieren Sie die Schlüsselworte `END`, `THEN` und `DO`, sowie alle Semikola `;`.
- Finden Sie eine sinnvolle Syntax für *program*.

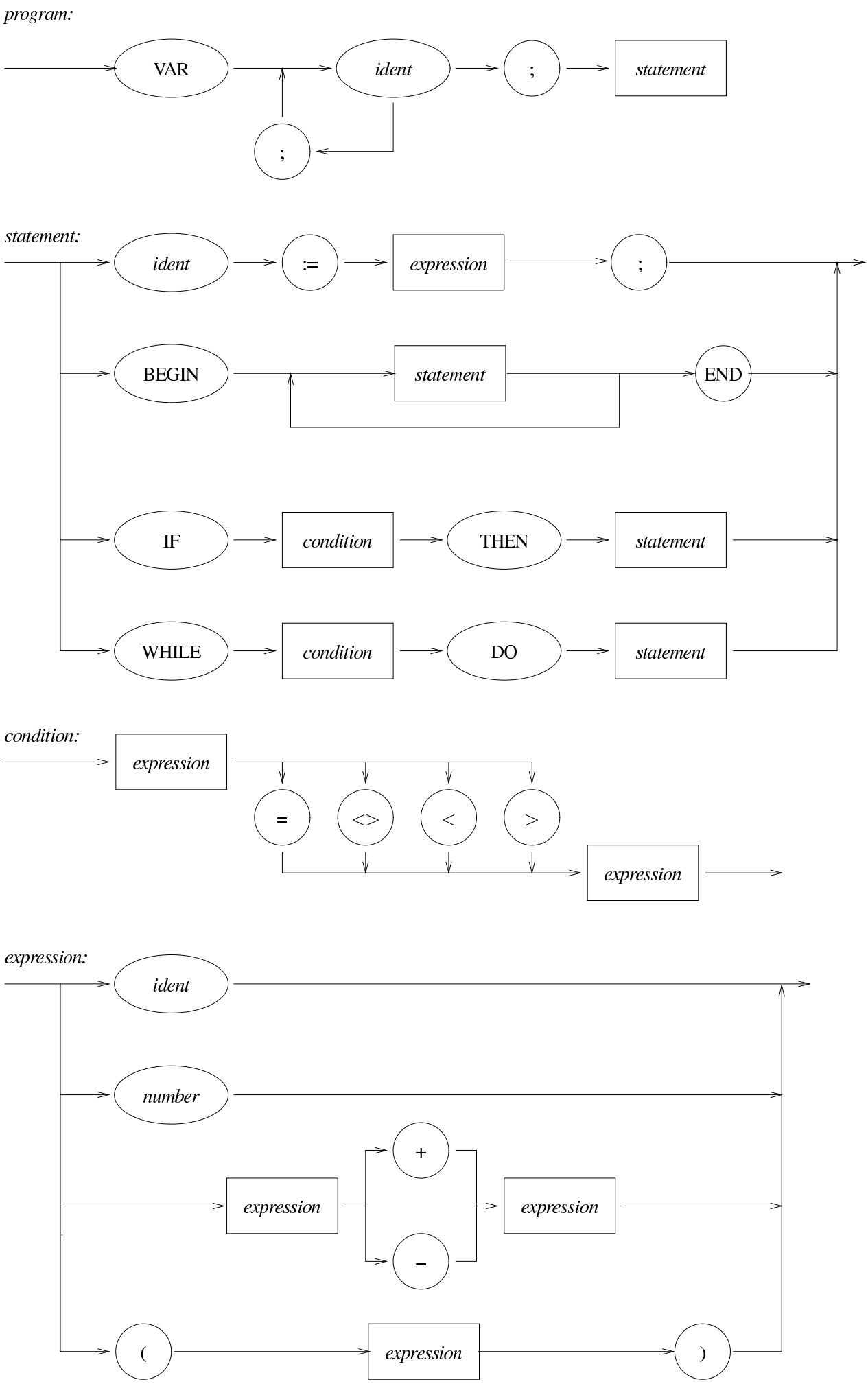


Abbildung 1: Syntax von P_0

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

5. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 26.05.2004, vor der Frontalübung

Aufgabe 14 (1+1+2 Punkte)

Eine Grammatik G sei durch folgenden Produktionen gegeben:

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

a) Eliminieren Sie die Linksrekursion mittels des in der Vorlesung angegebenen Verfahrens. Ist die resultierende Grammatik in $\mathrm{LL}(1)$?

b) Betrachten Sie die durch die folgenden Regeln gegebene Grammatik G':

$$\begin{array}{lcl} S & \rightarrow & S'\,S \mid S' \\ S' & \rightarrow & a \mid (\,T'\,) \\ T' & \rightarrow & S'\,T' \mid S' \end{array}$$

Führen Sie eine Linksfaktorisierung für G' durch.

c) Bestimmen Sie die *lookahead*–Mengen der Regeln für die in (b) erhaltene Grammatik. Ist sie aus $\mathrm{LL}(1)$?

Aufgabe 15 (2+2 Punkte)

Die kontextfreie Grammatik G sei gegeben durch:

$$\begin{array}{lcl} S & \rightarrow & AS \mid Ab \mid \varepsilon \\ A & \rightarrow & SA \mid a \end{array}$$

a) Geben Sie für das Wort *aaba* eine Berechnung einer Endkonfiguration des nichtdeterministischen BU-Analyseautomaten $\mathrm{NBA}(G)$ an.

b) Berechnen Sie die $\mathrm{LR}(0)$–Informationen zu G. Ist $G \in \mathrm{LR}(0)$?

Aufgabe 16 (4 Punkte)

Gegeben sei die folgende LL(1)–Grammatik G für *s-expressions* in EBNF:

$$
\begin{array}{lcl}
S & \rightarrow & SExpr^{+} \\
SExpr & \rightarrow & Atom \mid \texttt{(}\ SList\ \texttt{)} \\
SList & \rightarrow & SExpr\ SList \mid \varepsilon \\
Atom & \rightarrow & identifier \mid boolean \mid integer \mid float \mid string
\end{array}
$$

Erstellen Sie in C einen *Recursive Descent*–Parser mit *lookahead*–Mengen, der $L(G)$ erkennt. Der Parser soll als Eingabe einen Tokenstrom erhalten, der vom Lexer aus Aufgabe 10 erzeugt wird. Die Ausgabe soll die Analyse in Form einer Sequenz der Regelnummern (oder ein Fehler) sein.

- Schreiben Sie die Funktionen `void next_symbol()` und `token_t lookahead_symbol()`, die ein neues Symbol vom Lexer anfordern bzw. das aktuelle Symbol zurückliefern.
- Nehmen Sie die entsprechenden Änderungen an der `main()`–Funktion des Lexers vor, um den Parser aufzurufen.
- Verwenden Sie im Fehlerfall des Parsers die *location*–Informationen der Token in den Fehlermeldungen. Ebenfalls soll das Symbol ausgegeben werden, welches zum Fehler geführt hat, sowie die aktuelle *lookahead*–Menge, also die Menge der *erwarteten* Token.

Schicken Sie das zu erstellende Programm *zusätzlich* zur schriftlichen Form auch per Email an

`rieger+cb2004@i2.informatik.rwth-aachen.de`

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

6. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 09.06.2004, vor der Frontalübung

Aufgabe 17 (4 Punkte)

Für reduzierte kontextfreie Grammatiken ist die starke LL(k)–Eigenschaft (SLL(k)) wie folgt definiert:

Für $G \in CFG$, G reduziert, gilt:
$G \in \mathrm{SLL}(k) :\leftrightsquigarrow$ Für alle Regelpaare $A \to \beta \mid \gamma$ mit $\beta \neq \gamma$ gilt:

$$\underline{\mathrm{first}_k}(\beta\,\underline{\mathrm{follow}_k}(A)) \cap \underline{\mathrm{first}_k}(\gamma\,\underline{\mathrm{follow}_k}(A)) = \emptyset$$

Für $k = 1$ zeigt ein Satz der Vorlesung, daß $\mathrm{SLL}(1) = \mathrm{LL}(1)$ gilt.

Zeigen Sie am Beispiel der folgenden Grammatik, daß $\mathrm{SLL}(2) \neq \mathrm{LL}(2)$ gilt:

$$\begin{aligned} S &\to aAab \mid bAbb \\ A &\to a \mid \varepsilon \end{aligned}$$

Aufgabe 18 (2+2+2 Punkte)

Definition: Eine Grammatik heißt *nicht–trivial*, falls sie eine unendliche Sprache erzeugt.

a) Geben Sie eine nicht–triviale startseparierte Grammatik an, die zwar LL(1), aber nicht LR(0) ist.

b) Geben Sie eine nicht–triviale startseparierte Grammatik an, die zwar LR(0), aber nicht LL(1) ist.

c) Geben Sie eine nicht–triviale startseparierte Grammatik an, die sowohl LL(1) als auch LR(0) ist.

Geben Sie jeweils kurze Begründungen an.

Bemerkung: War Ihre Suche in allen drei Fällen erfolgreich, so haben Sie damit gezeigt, daß die Grammatikklassen LL(1) und LR(0) schief zueinander liegen und nicht disjunkt sind.

Aufgabe 19 (2+1+1 Punkte)

Betrachten Sie eine Grammatik $G(D_1)$ der Dyck-Sprache D_1 (die Sprache der korrekt geklammerten Wörter mit nur einem Klammertyp):

$$S \rightarrow (S)S \mid \varepsilon$$

a) Geben Sie die LR(0)–Mengen zu $G(D_1)$ an.

b) Geben Sie allgemeine Kriterien an, mit denen sich alleine anhand der LR(0)–Mengen entscheiden läßt, ob für eine gegebene Grammatik G gilt $G \in \mathrm{LR}(0)$ oder nicht.

c) Gilt $G(D_1) \in \mathrm{LR}(0)$? Begründen Sie Ihre Antwort.

Aufgabe 20 (1+2+1+1 Punkte)

Gegeben sei die Grammatik G mit folgenden Produktionen:

$$\begin{aligned} S &\rightarrow (L) \mid a \\ L &\rightarrow LS \mid S \end{aligned}$$

a) Konstruieren Sie den Automaten $\mathfrak{A}(G) \in \mathrm{NFA}$, der zur Berechnung der LR(0)–Mengen dient.

b) Berechnen Sie die LR(0)-Mengen, indem Sie die Potenzmengen-Konstruktion auf $\mathfrak{A}(G)$ anwenden.

c) Geben Sie die LR(0)–Analysetabelle für G an.

d) Geben Sie eine Berechnung des $\mathrm{DBA}(G)$ an für das Wort $w = (a(aa))$

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

7. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 16.06.2004, vor der Frontalübung

Aufgabe 21 (3 Punkte)

Konstruieren Sie die SLR(1)-Analysetabelle zu der Grammatik G mit den Regeln:

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & A \mid b \\ A & \rightarrow & Aa \mid Sb \end{array}$$

Aufgabe 22 (2+2 Punkte)

Gegeben sei die startseparierte Grammatik $G \notin \mathrm{LR}(0)$:

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & aAd \mid bBd \mid aBe \mid bAe \\ A & \rightarrow & c \\ B & \rightarrow & c \end{array}$$

a) Begründen Sie anhand der LR(0)-Mengen von G, warum $G \notin \mathrm{SLR}(1)$.

b) Prüfen Sie anhand der LR(1)-Mengen von G, ob $G \in \mathrm{LR}(1)$.

Aufgabe 23 (2+2+1 Punkte)

Gegeben sei die folgende kontextfreie Grammatik G:

$$\begin{array}{lcll} S' & \rightarrow & S & (0) \\ S & \rightarrow & AA & (1) \\ A & \rightarrow & aA \mid b & (2,3) \end{array}$$

a) Konstruieren Sie die Menge der LALR(1)–Informationen zu G sowie die zugehörige **goto**–Funktion.

b) Bestimmen Sie die LALR(1)–**action**–Funktion zu G.

c) Geben Sie die Konfigurationenfolge an, die der deterministische LALR(1)–Bottom–up–Analyse-automat nach Eingabe von $w = bab$ durchläuft.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

8. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 23.06.2004, vor der Frontalübung

Aufgabe 24 (2+2 Punkte)

Betrachten Sie noch einmal die LR(1)-Mengen der Grammatik $G \in \mathrm{LR}(1)$ aus Aufgabe 22:

$$\begin{aligned} S' &\to S \\ S &\to aAd \mid bBd \mid aBe \mid bAe \\ A &\to c \\ B &\to c \end{aligned}$$

a) Begründen Sie ausführlich, warum $G \notin \mathrm{LALR}(1)$.

b) Betrachten Sie alle möglichen Situationen, die bei der Vereinigung von zwei LR(0)-äquivalenten LR(1)-Mengen einer Grammatik $G \in (\mathrm{LR}(1) \setminus \mathrm{LALR}(1))$ auftreten können. Welche davon können potentiell zu einem Konflikt führen und welche nicht? Begründen Sie Ihre Antworten.

Aufgabe 25 (4 Punkte)

Geben Sie eine Attributgrammatik zur Berechnung des Wertes von konstanten arithmetischen Ausdrücken an. Benutzen Sie folgende kontextfreie Grammatik:

$$\begin{aligned} E' &\to E \\ E &\to E\ AO\ T \mid T \\ AO &\to \texttt{+} \mid \texttt{-} \\ T &\to T\ MO\ F \mid F \\ MO &\to \texttt{*} \mid \texttt{/} \\ F &\to \texttt{(}\ E\ \texttt{)} \mid \texttt{realconst} \end{aligned}$$

Dabei sei `realconst` ein Terminalsymbol mit einem synthetischen Attribut v, welches den Wert der Konstanten enthält.

Ermitteln Sie den Ableitungsbaum für die Eingabe `(3+5.2*7)/3.4`, stellen Sie das zugehörige Attributgleichungssystem auf und lösen Sie es.

Aufgabe 26 (4 Punkte)

Geben Sie eine Attributgrammatik an, die die Transformation eines vollständig geklammerten Booleschen Ausdrucks in disjunktive Normalform (DNF) beschreibt. Die Ausdrücke seien aus Variablenbezeichnern `ID`, Klammersymbolen `(,)` und den Junktoren $\wedge$, $\vee$ und $\neg$ aufgebaut. Das Symbol `ID` habe ein synthetisches Attribut $id \in ID$ mit dem Namen des Bezeichners.

Ermitteln Sie den Ableitungsbaum für die Eingabe `((x` $\vee$ `y)` $\wedge$ $\neg$`(x` $\vee$ $\neg$`y))`, stellen Sie das zugehörige Attributgleichungssystem auf und lösen Sie es.

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

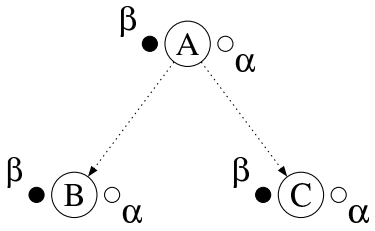
LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

9. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 30.06.2004, vor der Frontalübung

Aufgabe 27 (3 Punkte)

Geben Sie für die unten angegebene Attributzuordnung zu der Produktion $\pi = A \rightarrow BC$ einen Abhängigkeitsgraphen DG_π an, der alle erlaubten Abhängigkeiten von inheriten und synthetischen Attributen enthält.



Aufgabe 28 (1+3+2 Punkte)

Betrachten Sie die folgende Attributgrammatik G mit $\alpha_i \in \underline{\text{Syn}}$ und $\beta_j \in \underline{\text{Inh}}$:

$$\pi_1 = S' \rightarrow A \qquad \alpha_3.0 = \alpha_2.1 \qquad \pi_2 = A \rightarrow aA \qquad \begin{array}{l} \beta_2.1 = \beta_1.0 \\ \beta_1.1 = \alpha_1.1 \\ \alpha_1.0 = \alpha_2.1 \end{array}$$

$$\begin{array}{ll} \pi_3 = A \rightarrow a & \alpha_2.0 = \beta_1.0 \\ \pi_4 = A \rightarrow b & \alpha_1.0 = \beta_2.0 \end{array}$$

a) Zeichnen Sie die Abhängigkeitsgraphen DG_{π_i}.

b) Untersuchen Sie mit Hilfe des in der Vorlesung vorgestellten Verfahrens, ob G zirkulär ist.

c) Ist G stark nicht-zirkulär?

Hinweis: Geben Sie die Attributabhängigkeitsmengen D jeweils graphisch an!

Aufgabe 29 (4 Punkte)

Betrachten Sie noch einmal die EBNF-Grammatik G aus Aufgabe 16:

$$
\begin{array}{lcl}
S & \rightarrow & SExpr^{+} \\
SExpr & \rightarrow & Atom \mid \texttt{(}\ SList\ \texttt{)} \\
SList & \rightarrow & SExpr\ SList \mid \varepsilon \\
Atom & \rightarrow & identifier \mid boolean \mid integer \mid float \mid string
\end{array}
$$

Schreiben Sie unter Verwendung von `yacc` oder `bison` einen Parser für G, der äquivalente Ergebnisse zu dem *recursive descent*-Parser aus Aufgabe 16 liefert.

Insbesondere sollen folgende Punkte behandelt werden:

- Verwenden Sie den Dateinamen `utsl.y` für die Parserspezifikation.
- Sehen Sie aussagekräftige Fehlermeldungen vor, ähnlich wie in Aufgabe 16.
- In einer *action*-Regel der Parserspezifikation soll als semantische Aktion die Nummer der reduzierten Regel ausgegeben werden.
- Dokumentation des Quellcodes und sinnvolle Testläufe (auch für Fehlerfälle) werden ebenfalls wieder erwartet.
- Schicken Sie das zu erstellende Programm *zusätzlich* zur schriftlichen Form auch per Email an

  `rieger+cb2004@i2.informatik.rwth-aachen.de`

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

10. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 07.07.2004, vor der Frontalübung

Aufgabe 30 (4 Punkte)

Untersuchen Sie mit Hilfe des in der Vorlesung vorgestellten Verfahrens, ob folgende Attributgrammatik zirkulär ist:

$S \longrightarrow A \quad \beta.1 = \alpha.1$

$A \longrightarrow AB \quad \alpha.0 = \alpha.1,\ \beta.2 = \beta.0,\ \beta.1 = \alpha.2$

$A \longrightarrow a \quad \alpha.0 = \beta.0$

$B \longrightarrow b \quad \alpha.0 = \beta.0$

Aufgabe 31 (3 Punkte)

Erweitern Sie den Parser aus Aufgabe 29, so daß er einen abstrakten Syntaxbaum (AST) zur Eingabe berechnet und diesen ausgibt. Dokumentieren Sie die Struktur des AST mit Hilfe der Typsignaturen seiner Knotenkonstruktoren (Op.-Symbole).

Hinweis:

- Verwenden Sie elementare Datenstrukturen wie z. B. einfach verkettete Listen zur Darstellung des Baumes auf dem Heap.
- Schicken Sie das zu erstellende Programm *zusätzlich* zur schriftlichen Form auch per Email an

`rieger+cb2004@i2.informatik.rwth-aachen.de`

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

11. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 14.07.2004, vor der Frontalübung

Aufgabe 32 (4+2 Punkte)

Gegeben sei die Grammatik G mit

$$
\begin{array}{lll}
\mathit{SExpr} & \rightarrow & \texttt{nil} \\
 & | & \texttt{(setf } \mathit{identifier}\ \mathit{SExpr}\texttt{)} \\
 & | & \texttt{(if } \mathit{SExpr}\ \mathit{SExpr}\ \mathit{SExpr}\texttt{)} \\
 & | & \texttt{(while } \mathit{SExpr}\ \mathit{SList}\texttt{)} \\
 & | & \texttt{(let (}\mathit{LetDefs}\texttt{)} \ \mathit{SList}\texttt{)} \\
 & | & \texttt{(}\mathit{SList}\texttt{)} \\
 & | & \mathit{Atom} \\
\mathit{SList} & \rightarrow & \mathit{SExpr}\ \mathit{SList} \mid \varepsilon \\
\mathit{LetDefs} & \rightarrow & \texttt{(}\mathit{identifier}\ \mathit{SExpr}\texttt{)}\ \mathit{LetDefs} \mid \varepsilon \\
\mathit{Atom} & \rightarrow & \mathit{identifier} \mid \mathit{boolean} \mid \mathit{integer} \mid \mathit{float}
\end{array}
$$

a) Erweitern Sie die Grammatik G um ein Attributschema, mit dessen Hilfe der Typ $t \in T$ eines Ausdrucks als synthetisches Attribut *type* abgelesen werden kann.

Dabei gelte:

- Die Menge der Typen sei gegeben durch $T := T_0 \cup T_l \cup T_f \cup \{\texttt{null}, \texttt{error}\}$ mit:

$$
\begin{array}{lr}
T_0 := \{\texttt{float}, \texttt{integer}, \texttt{boolean}\} & \text{Basistypen} \\
T_l := \{\texttt{lval}\ t \mid t \in T_0\} & \textit{location}\text{-Typen} \\
T_f := T^+ \rightarrow T & \text{Funktionstypen}
\end{array}
$$

statements haben den Ergebnistyp `null` (*unit*-Typ), der Typ `error` symbolisiert einen Typfehler.

- Die Typen einiger Funktionen seien durch folgende Typschemata vordefiniert:

$$
\begin{array}{ll}
\texttt{nil} & : \texttt{null} \\
\texttt{setf} & : \forall t \in T_0.\ \texttt{lval}\ t \times t \rightarrow \texttt{null} \\
\texttt{if} & : \forall t \in T.\ \texttt{boolean} \times t \times t \rightarrow t \\
\texttt{while} & : \forall t_i \in T.\ \texttt{boolean} \times t_1 \times \cdots \times t_n \rightarrow \texttt{null} \\
\texttt{let} & : \forall t_i, t'_j \in T.\ ((\texttt{lval}\ t_1 \times t_1) \times \cdots \times (\texttt{lval}\ t_n \times t_n)) \times t'_1 \times \cdots \times t'_m \rightarrow t'_m \\
= & : \forall t \in T_0.\ t \times t \rightarrow \texttt{boolean} \\
< & : \forall t \in \{\texttt{integer}, \texttt{float}\}.\ t \times t \rightarrow \texttt{boolean} \\
*, + & : \forall t_i \in \{\texttt{integer}, \texttt{float}\}.\ t_1 \times t_2 \rightarrow \mathit{max}\{t_1, t_2\}
\end{array}
$$

- Das Terminalsymbol *identifier* besitzt ein synthetisches Attribut $ident \in IDENT$, das den Namen des Bezeichners enthält.
- Der Algorithmus zur Typberechnung umfaßt folgende Regeln:
  - Ein Atom *integer* hat den Typ `integer`, etc.
  - Bezüglich *max* gilt: `integer` $<$ `float`.
  - Bei der Deklaration einer Variablen v (mittels `let`) wird die Funktion, welche die Symboltabelle repräsentiert, punktweise erweitert: $symtab(v) := t$, wobei t der Typ des v zugewiesenen Ausdrucks ist.
  - Falls eine Funktion ein Argument des Typs `lval` t verlangt, und der Variablenbezeichner v der aktuelle Parameter des Ausdrucks an dieser Stelle ist, dann soll dessen Typ `lval` t' sein, falls $t' = symtab(v)$, ansonsten `error`.
  - Die aktuellen Parameter eines Funktionsaufrufs müssen in korrekter Anzahl und mit passenden Typen auftreten, ansonsten soll der Typ des Ausdrucks auf `error` gesetzt werden.
  - Hat ein Teilausdruck den Typ `error`, so hat auch der gesamte Ausdruck den Typ `error`.

  Hinweise:
  - Benutzen Sie ein inherites Attribut $symtab : IDENT \rightarrow T$, welchem die Symboltabelle in Form einer Funktion zugewiesen wird.
    Eine solche Funktion weist einem Bezeichner $ident \in IDENT$ seinen Typ $symtab(ident)$ zu.

b) Stellen Sie das Attributgleichungssystem für den folgenden Ausdruck auf und lösen Sie es:

```
(let ((x 0)
      (y 10)
      (z 1024))
  (while (< x y)
     (let ((y (* 0.5 z)))
       (setf x (+ 1 x))
       (setf z y)))
  z)
```

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

12. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 21.07.2004, vor der Frontalübung

Aufgabe 33 (3 Punkte)

Gegeben sei das BPS–Programm $P \in \underline{\text{Prog}}^{(1)}$

in/**out** Y;
const $X = 4$;
while $(Y * Y) < X$ **do** $Y := Y + 1$

Berechnen Sie die Programmsemantik $\mathfrak{M}[\![P]\!](1)$.

Aufgabe 34 (2 Punkte)

Geben Sie ein BPS-Programm $P \in \underline{\text{Prog}}^{(2)}$ an, das für eine Eingabe $N \in \mathbb{N}$ die N-te Fibonacci-Zahl F_N durch *Dynamisches Programmieren* berechnet. Es gelte $F_0 = 0, F_1 = 1$.

Aufgabe 35 (4 Punkte)

Programmieren Sie eine Funktion im *Zwischencode* für BPS, welche für eine Eingabe $N \in \mathbb{N}$ die N-te Fibonacci-Zahl berechnet.

Geben Sie je eine iterative (ohne die Befehle **CALL** und **RET**) und eine rekursive Lösung an.

Aufgabe 36 (4 Punkte)

Implementieren Sie eine abstrakte Maschine für den Zwischencode für BPS. Sie können dafür eine höhere Programmiersprache ihrer Wahl verwenden.

Schicken Sie das zu erstellende Programm *zusätzlich* zur schriftlichen Form auch per Email an

`rieger+cb2004@i2.informatik.rwth-aachen.de`

RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL FÜR INFORMATIK II
RWTH Aachen · D-52056 Aachen
http://www-i2.informatik.rwth-aachen.de/

Prof. Dr. K. Indermark
M. Weber

13. Übung zu „Compilerbau“, SS 2004
Abgabe: Mi., 28.07.2004, vor der Frontalübung

Aufgabe 37 (2+4+3 Punkte)

Gegeben sei das Programm $P \in \underline{\text{Prog}}^{(1)}$

```
in/out X,Y;
    var Z;
    proc G;
        var Z;
        Z := 1;
        if (Y = 0) then Y := X;
        else begin
            Z := X mod Y;
            X := Y;
            Y := Z;
            G();
        end;
    begin
    Z := 0;
    G();
    X := Z;
    end.
```

a) Erweitern Sie die BPS-Programmiersprache um den Operator **mod** (Modulus, x **mod** y):

- Passen Sie dazu die Ausdruckssemantik $\mathfrak{E}_b[\![\cdot]\!]\rho\sigma$ an.
- Erweitern Sie die Befehlssemantik $[\![B]\!] : ZR \dashrightarrow ZR$ für **mod**.
- Erweitern Sie die Übersetzungsfunktion $\underline{\text{trans}}$: BPS $\rightarrow$ AM-Code entsprechend. Gehen Sie davon aus, daß ein Maschinen-Befehl `MOD` existiert.

b) Ermitteln Sie **trans**(P).

c) Berechnen Sie die Semantik $\mathfrak{I}[\![\textbf{trans}(P)]\!]$ für die Eingaben $X := 4$, $Y := 2$.

Scheinklausur

Falls Sie an der Scheinklausur teilnehmen möchten, melden Sie sich bitte bis

Freitag, 23.07.2004,

mit einer Email (Subject: `Klausuranmeldung` *`Name Vorname Matrikelnr.`*) an

`rieger+klausur-cb2004@i2.informatik.rwth-aachen.de`.

Auf den Webseiten wird dann am Montag, 26.07.2004, eine Liste mit Matrikelnummern derjenigen, die sich angemeldet haben, veröffentlicht.

Falls Sie sich nicht sicher sind, ob Sie die Voraussetzungen erfüllen, melden Sie sich trotzdem an.