



## Übungsblatt 9 – Lösungsvorschläge

14.12.2001

### Aufgabe 1 SLL(1)-Grammatik der while-Sprache

(Schuster)

Sie haben die einfache while-Sprache ohne Erweiterungen kennengelernt.

Überführen Sie die Grammatik in SLL(1) und notieren Sie diese in *EBNF*-Form. Um die Grammatik später mit einem Parser-Generator einzulesen, sollten Sie alle Sonderzeichen „≠“, „:=“, etc. durch Schlüsselwörter „isnot“, „becomes“, etc. ersetzen oder ersatzlos entfernen.

Betrachten wir die Produktionen und deren  $First_1(sFollow_1(X))$  Mengen

<Ziffer>	::= "0"   "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9" .	{0, 1, ...}
<Zahl>	::= <Ziffer>	{0, 1, ...}
	<Zahl> <Ziffer> .	{0, 1, ...}
<Variable>	::= "X" <Zahl> .	{X}
<Term>	::= "0"	{0}
	"Vorg(" <Variable> ")"	{V}
	"Nachf(" <Variable> ")"	{N}
	"(" <Term> ")" .	{(}
<Prädikat>	::= <Variable> "≠" <Variable> .	{X}
<Anweisung>	::= <Variable> ":= " <Term>	{X}
	"while" <Prädikat> "do" <AnwFolge> "od"	{w}
	<Programm> .	{b}
<AnwFolge>	::= <Anweisung>	{x, w, b}
	<AnwFolge> ";" <Anweisung> .	{x, w, b}
<Programm>	::= "begin" "end"	{b}
	"begin" <AnwFolge> "end" .	{b}

In der letzten Spalte sind die  $First_1(sFollow_1(X))$  aufgeführt. In drei Fällen sind die Schnittmengen der alternativen Produktionen nicht leer.

Die Definition einer Zahl erfolgt in Form einer Linkrekursion, welche in eine Rechtsrekursion überführt werden muß:

Aus

$$\langle \text{Zahl} \rangle ::= \langle \text{Ziffer} \rangle \mid \langle \text{Zahl} \rangle \langle \text{Ziffer} \rangle .$$

wird dabei

$$\langle \text{Zahl} \rangle ::= \langle \text{Ziffer} \rangle \langle \text{Zahl}' \rangle . \text{ und}$$
$$\langle \text{Zahl}' \rangle ::= \varepsilon \mid \langle \text{Ziffer} \rangle \langle \text{Zahl}' \rangle .$$

Das gleiche Problem liegt auch bei einer Anweisungsfolge vor:

Aus

$$\langle \text{AnwFolge} \rangle ::= \langle \text{Anweisung} \rangle \mid \langle \text{AnwFolge} \rangle \text{";"} \langle \text{Anweisung} \rangle .$$

wird

$$\langle \text{AnwFolge} \rangle ::= \langle \text{Anweisung} \rangle \langle \text{AnwFolge}' \rangle .$$
$$\langle \text{AnwFolge}' \rangle ::= \varepsilon \mid \text{";"} \langle \text{Anweisung} \rangle \langle \text{AnwFolge}' \rangle .$$

Die letzte Umformung muß für Programme durchgeführt werden. Bei den Alternativen liegt ein gemeinsames Präfix vor, welches „herausgezogen“ werden muß.

Aus

$$\langle \text{Programm} \rangle ::= \text{"begin"} \text{"end"} \mid \text{"begin"} \langle \text{AnwFolge} \rangle \text{"end"} .$$

wird

$$\langle \text{Programm} \rangle ::= \text{"begin"} \langle \text{Programm}' \rangle$$
$$\langle \text{Programm}' \rangle ::= \text{"end"} \mid \langle \text{AnwFolge} \rangle \text{"end"} .$$

## Aufgabe 2 SLL(1), Zerteilertabelle

(Bierwald)

Sprachschatz der folgenden Grammatik  $G = (N, T, \Pi, S)$  ist eine Sprache aussagenlogischer Formeln.

$S = \text{Formel}$

$N = \{\text{Formel}, \text{Atom}, \text{Negation}, \text{Konjunktion}, \text{Disjunktion}\}$

$T = \{P, ', \sim, \&, +, (, )\}$

Formel ::= Atom | Negation | Konjunktion | Disjunktion

Atom ::= Atom ' | P

Negation ::= ~ Formel

Konjunktion ::= (Formel & Formel)

Disjunktion ::= (Formel + Formel)

a) Die Grammatik ist nicht SLL(1). Zeigen Sie alle Stellen auf, an denen die SLL(1)-Eigenschaft verletzt wird.

1. Stelle (Gemeinsamer Präfix): Formel ::= Konjunktion | Disjunktion

$$\text{First}_1(\text{Konjunktion Follow}(\text{Formel})) = \{($$
$$\text{First}_1(\text{Disjunktion Follow}(\text{Formel})) = \{($$

2. Stelle (Linksrekursion): Atom ::= Atom ' | P

$$\text{First}_1(\text{Atom ' Follow}(\text{Atom})) = \{P\}$$
$$\text{First}_1(P \text{ Follow}(\text{Atom})) = \{P\}$$

b) Modifizieren Sie die Grammatik so, dass sie SLL(1) ist, beweisen Sie dies und stellen anschließend die Zerteilertabelle auf.

Es gibt viele Möglichkeiten. Eine davon ist die folgende: Wir führen drei zusätzliche Nichtterminale *Hochkommas*, *BinOp* und *KonDis* ein und betrachten die Produktionen

Formel ::= Atom | Negation | KonDis  
 Atom ::= P Hochkommas  
 Hochkommas ::= ‘ Hochkommas | ε  
 Negation ::= ~ Formel  
 KonDis ::= (Formel BinOp Formel)  
 BinOp ::= & | +

$\text{First}_1(\text{Atom Follow}(\text{Formel})) = \{P\}$

$\text{First}_1(\text{Negation Follow}(\text{Formel})) = \{\sim\}$

$\text{First}_1(\text{KonDis Follow}(\text{Formel})) = \{(\}$

Schnitte sind paarweise disjunkt.

$\text{First}_1(\text{‘ Hochkommas Follow}(\text{Hochkommas})) = \{\text{‘}\}$

$\text{First}_1(\text{ε Follow}(\text{Hochkommas})) = \text{First}_1(\text{Follow}(\text{Hochkommas})) = \{\&, +, ), \#\}$

Schnitt disjunkt.

$\text{First}_1(\& \text{ Follow}(\text{BinOp})) = \{\&\}$

$\text{First}_1(+ \text{ Follow}(\text{BinOp})) = \{+\}$

Schnitt disjunkt.

Grammatik ist also SLL(1).

	P	‘	~	&
Formel	Formel q ::= Atom q	∅	Formel q ::= Negation q	∅
Atom	Atom q ::= Hochkommas P q	∅	∅	∅
Hochkommas	∅	Hochkommas q ::= Hochkommas ‘ q	∅	Hochkommas q ::= q
Negation	∅	∅	Negation q ::= Formel ~ q	∅
KonDis	∅	∅	∅	∅
BinOp	∅	∅	∅	BinOp q ::= & q

  

	+	(	)	#
Formel	∅	Formel q ::= KonDis q	∅	∅
Atom	∅	∅	∅	∅

Hochkommas	Hochkommas $q ::= q$	$\emptyset$	Hochkommas $q ::= q$	Hochkommas $q ::= q$
Negation	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
KonDis	$\emptyset$	KonDis $q ::=$ ) Formel BinOp Formel ( $q$	$\emptyset$	$\emptyset$
BinOp	BinOp $q ::=$ + $q$	$\emptyset$	$\emptyset$	$\emptyset$

### Aufgabe 3 Prädikate

(Bierwald)

Zeigen Sie, daß  $X < Y$  genau dann gilt, wenn  $(Y \dot{-} X) \dot{-} Vorg(Y \dot{-} X) = 1$  ist.

*Links nach Rechts:*

Sei  $x < y$ .

Daraus folgt:  $y \dot{-} x = y - x$

Daraus folgt:  $(y \dot{-} x) \dot{-} Vorg(y \dot{-} x) = (y - x) \dot{-} Vorg(y - x)$

Für alle  $z$  gilt  $Vorg(z) \leq z$ , daher ist  $(y - x) \dot{-} Vorg(y - x) = (y - x) - Vorg(y - x)$

Wegen  $x < y$  ist  $y - x > 0$  und damit  $Vorg(y - x) = y - x - 1$ .

Also:  $(y \dot{-} x) \dot{-} Vorg(y \dot{-} x) = (y - x) - Vorg(y - x) = (y - x) - (y - x - 1) = 1$ .

Andere Richtung: Übung.

### Aufgabe 4 While-Programme

(Käufel)

Schreiben Sie while-Programme ohne Erweiterungen, welche die modifizierte Differenz, die ganzzahlige Division *div* und *mod*, den Rest der ganzzahligen Division zweier Zahlen, bestimmen.

Addition

Abkürzung:  $x \leftarrow X_1, y \leftarrow X_2, z \leftarrow X_3$

$z := 0;$

**while**  $y \neq z$  **do**  $x := Nachf(x); y := Vorg(y)$  **od**

Modifizierte Differenz

Abkürzung:  $x \leftarrow X_1, y \leftarrow X_2, n \leftarrow X_3$

$x - y = x - y - (y - y)$

$n := 0;$  **while**  $y \neq n$  **do**  $x := Vorg(x); y := Vorg(y)$  **od**

Multiplikation

Abkürzung:  $x \leftarrow X_1, y \leftarrow X_2$ . Die übrigen Variablen können beliebig durch solche der Form  $X_i$  ersetzt werden.

$z := 0; n := 0;$

**while**  $y \neq n$  **do**

$$z := z + x; \quad \left\{ \begin{array}{l} z_1 := 0; y_1 := \text{Nachf}(x); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y_1 \neq z_1 \ \mathbf{do} \ z := \text{Nachf}(z); y_1 := \text{Vorg}(y_1) \ \mathbf{od} \end{array} \right.$$

$y := \text{Vorg}(y)$   
**od;**  
 $x := \text{Nachf}(z);$   
 $x := \text{Vorg}(z)$

### Ganzzahlige Division

$$z := 0; \mathbf{while} \ y \leq x \ \mathbf{do} \ x := x \dot{-} y; z := z + 1 \ \mathbf{od}; x := z$$

$y \leq x$  gilt genau dann, wenn  $\neg x < y$ . An Stelle der in der Vorlesung angegebenen Ersetzung der Negation testen wir, in der While-Schleife, ob die charakteristische Funktion  $(y \dot{-} x) \dot{-} \text{Vorg}(y \dot{-} x)$  von  $x < y$  einen von 1 verschiedenen Wert hat. In diesem Fall gilt  $\neg x < y$ .

Abkürzung:  $x \leftarrow X_1, y \leftarrow X_2$ .

$z := 0; e := 1;$

$$h_1 := y \dot{-} x; \quad \left\{ \begin{array}{l} n := 0; \\ h_1 := \text{Nachf}(y); h_1 := \text{Vorg}(h_1); y_1 := \text{Nachf}(x); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od}; \end{array} \right.$$

$h_1 := h_1 \dot{-} \text{Vorg}(h_1); y_1 := \text{Vorg}(h_1); \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od};$

**while**  $h_1 \neq e$  **do**

$$x := x \dot{-} y; \quad \left\{ \begin{array}{l} y_1 := \text{Nachf}(y); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y \neq n \ \mathbf{do} \ x := \text{Vorg}(x); y_1 := \text{Vorg}(y_1) \ \mathbf{od} \end{array} \right.$$

$z := \text{Nachf}(z);$

$$h_1 := y \dot{-} x; \quad \left\{ \begin{array}{l} h_1 := \text{Nachf}(y); h_1 := \text{Vorg}(h_1); y_1 := \text{Nachf}(x); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od}; \end{array} \right.$$

$h_1 := h_1 \dot{-} \text{Vorg}(h_1) \quad y_1 := \text{Vorg}(h_1); \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od};$

**od;**

$x := \text{Nachf}(z);$

$x := \text{Vorg}(z);$

### Divisionsrest

$$\mathbf{while} \ y \leq x \ \mathbf{do} \ x := x \dot{-} y \ \mathbf{od}$$

Abkürzung:  $x \leftarrow X_1, y \leftarrow X_2$ .

$e := 1;$

$$\begin{array}{l}
h_1 := y \dot{-} x; \quad \left\{ \begin{array}{l} n := 0; \\ h_1 := \text{Nachf}(y); h_1 := \text{Vorg}(h_1); y_1 := \text{Nachf}(x); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od}; \end{array} \right. \\
h_1 := h_1 \dot{-} \text{Vorg}(h_1); \quad y_1 := \text{Vorg}(h_1); \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od}; \\
\mathbf{while} \ h_1 \neq e \ \mathbf{do} \\
\quad \left\{ \begin{array}{l} y_1 := \text{Nachf}(y); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y \neq n \ \mathbf{do} \ x := \text{Vorg}(x); y_1 := \text{Vorg}(y_1) \ \mathbf{od} \end{array} \right. \\
h_1 := y \dot{-} x; \quad \left\{ \begin{array}{l} h_1 := \text{Nachf}(y); h_1 := \text{Vorg}(h_1); y_1 := \text{Nachf}(x); y_1 := \text{Vorg}(y_1); \\ \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od}; \end{array} \right. \\
h_1 := h_1 \dot{-} \text{Vorg}(h_1) \quad y_1 := \text{Vorg}(h_1); \mathbf{while} \ y_1 \neq n \ \mathbf{do} \ h_1 := \text{Vorg}(h_1); y_1 := \text{Vorg}(y_1) \ \mathbf{od}; \\
\mathbf{od}
\end{array}$$

### Aufgabe 5 Programmäquivalenz

(Käuf)

Zeigen Sie, daß die Bedingung „ $u, v$  neu“ bei der Umwandlung der If-Then-Else-Anweisung in eine Folge von zwei If-Then-Anweisungen notwendig ist.

Beweis durch geschickt gewähltes Beispiel.

```

X1 := Nachf(0);
X2 := 0;
if X1 ≠ X2 then X2 := Nachf(X2) else X1 := Nachf(X1) fi

```

Endwerte: X1 = 1, X2 = 1

```

X1 := Nachf(0);
X2 := 0;
if X1 ≠ X2 then X2 := Nachf(X2) fi;
if ¬X1 ≠ X2 then X1 := Nachf(X1) fi;

```

Endwerte: X1 = 2, X2 = 1