

Übungen zu Informatik I Wintersemester 03/04

Übungsleiter: Dipl.-Inform. Tom Gelhausen



- ÄÖÜäöüß – Teil des deutschen Alphabets?
 - Wenn Ihnen Aufgaben nicht ganz klar formuliert zu sein scheinen, schreiben Sie Ihre Interpretation der Aufgabe dazu
 - Entspricht Ihre Bearbeitung Ihrer (nachvollziehbaren!) Interpretation der Aufgabe und ist diese richtig, bekommen Sie Ihre Punkte
 - Lösungen, bei denen nur das Endergebnis („42“) verzeichnet, der Lösungsweg jedoch nicht ersichtlich ist, werden nicht akzeptiert
- ϵ im Zeichenvorrat?
 - ϵ beschreibt das leere Wort der Länge 0, gebildet aus 0 Zeichen
 - Welches der Zeichen aus dem Zeichenvorrat 0x genommen wird, um ϵ zu bilden ist egal
 - ϵ kann sogar aus einem leeren Zeichenvorrat gebildet werden (es werden ja keine Zeichen dafür benötigt)
 - ϵ ist als Wort nicht im Zeichenvorrat enthalten, kann jedoch mit jedem Zeichenvorrat gebildet werden



1.6 Chomsky-Hierarchie

3

CH-0 Grammatik:

(lies: Chomsky 0) alle Produktionen haben allgemeine Form $l \mapsto r$; $l, r \in \mathcal{V}^*$ beliebig

☛ berechenbar, ob $S \Rightarrow^* y$

☛ **aber nicht immer, ob $S \not\Rightarrow^* y$**

CH-1 Grammatik:

(auch kontextsensitive Grammatik)

alle Produktionen sind kontextsensitiv

(evtl. Ausnahme: $S \mapsto \varepsilon$, **dann aber nicht $\dots \mapsto \dots S \dots!$**)

Kontextsensitive Produktion:

$uAv \mapsto urv$; $A \in \mathcal{N}$; $r \in \mathcal{V}^+$; $u, v \in \mathcal{V}^*$

äquivalente Definition:

alle Produktionen sind beschränkt

Beschränkte Produktion:

$l \mapsto r$; $l, r \in \mathcal{V}^*$; $1 \leq |l| \leq |r|$

☛ In Ableitung $S \Rightarrow^* x \Rightarrow y$ gilt $|y| \geq |x|$

☛ Entscheidbar, ob $S \Rightarrow^* y$



- (A) $uAv \mapsto urv$; $A \in \mathcal{N}$; $r \in \mathcal{V}^+$; $u, v \in \mathcal{V}^*$
- (B) $l \mapsto r$; $l, r \in \mathcal{V}^*$; $1 \leq |l| \leq |r|$ z.B.: $cB \mapsto Bc$

	Grammatik	Sprache
(A)	G von Typ A \Rightarrow G von Typ B	-
(B)	G von Typ B $\not\Rightarrow$ G von Typ A	G von Typ B \Rightarrow \exists strukturäquivalente Grammatik G' mit G' vom Typ A und $\mathcal{L}(G) = \mathcal{L}(G')$ (Beweis im Buch S. 36)



- Chomsky Typ 2 Grammatiken beschreiben die Struktur korrekter Programme in Programmiersprachen wie Pascal, C, C#, oder Java
- Erinnerung:
 - $A \mapsto r$; $A \in \mathcal{N}$; $r \in \mathcal{V}^*$
 - Σ Menge der Terminale
 - \mathcal{N} Menge der Nichtterminale
 - $\mathcal{V} = \Sigma \cup \mathcal{N}$
- Hier am Beispiel der Sprache Java
- Spezifikation unter <http://java.sun.com/docs/books/jls/>



CHAPTER 2

Grammars

THIS chapter describes the context-free grammars used in this specification to define the lexical and syntactic structure of a program.

2.1 Context-Free Grammars

A context-free grammar consists of a number of productions. Each production has an abstract symbol called a nonterminal as its left-hand side, and a sequence of one or more nonterminal and terminal symbols as its right-hand side. For each grammar, the terminal symbols are drawn from a specified alphabet.

Starting from a sentence consisting of a single distinguished nonterminal, called the goal symbol, a given context-free grammar specifies a language, namely, the set of possible sequences of terminal symbols that can result from repeatedly replacing any nonterminal in the sequence with a right-hand side of a production for which the nonterminal is the left-hand side.



2.4 Grammar Notation

Terminal symbols are shown in `fixed width` font in the productions of the lexical and syntactic grammars, and throughout this specification whenever the text is directly referring to such a terminal symbol. These are to appear in a program exactly as written.

Nonterminal symbols are shown in *italic* type. The definition of a nonterminal is introduced by the name of the nonterminal being defined followed by a colon. One or more alternative right-hand sides for the nonterminal then follow on succeeding lines. For example, the syntactic definition:

```
IfThenStatement:  
    if ( Expression ) Statement
```

states that the nonterminal *IfThenStatement* represents the token `if`, followed by a left parenthesis token, followed by an *Expression*, followed by a right parenthesis token, followed by a *Statement*.



- Bekannte Notation (aus C, C#, Java, ...)

```
for (i=0; i<5; i++)          FOR i:=0 TO 4 DO
{                               BEGIN
    // do something usefull ...
}                               END
```

- Aber was macht man, wenn man das `i` nicht am Ende der Schleife, sondern mitten drin erhöhen will?

```
for (i=0; i<10; i=i+0)
{
    // do something with old value of i
    i++;
    // do something with new value of i
}
```



- *ForStatement*:

```
for ( ForInitopt ; Expressionopt ; ForUpdateopt ) { ... }
```

- *ForStatement*:

```
for ( ForInit ; Expression ; ForUpdate ) Statement
```

```
for ( ForInit ; Expression ; ) Statement
```

```
for ( ForInit ; ; ForUpdate ) Statement
```

```
for ( ForInit ; ; ) Statement
```

```
for ( ; Expression ; ForUpdate ) Statement
```

```
for ( ; Expression ; ) Statement
```

```
for ( ; ; ForUpdate ) Statement
```

```
for ( ; ; ) Statement
```

- The *Expression* must have type boolean, or a compile-time error occurs.



- *ForInit:*

StatementExpressionList
LocalVariableDeclaration

- *ForUpdate:*

StatementExpressionList

- *StatementExpressionList:*

StatementExpression
StatementExpressionList , *StatementExpression*

SE, SE, SE, SE

- *LocalVariableDeclarationStatement:*

LocalVariableDeclaration ;

- *LocalVariableDeclaration:*

final_{opt} *Type VariableDeclarators*

- *ExpressionStatement:*

StatementExpression ;

- *StatementExpression:*

Assignment

[...]

MethodInvocation

ClassInstanceCreationExpression

ForStatement:

for (*ForInit*_{opt} ; *Expression*_{opt} ; *ForUpdate*_{opt})
Statement



- Dementsprechend geht auch

```
for (int i=0, j=10; (i<10) && (j>0); i++, j++)  
{  
    // do something with i and j  
}
```

- oder

```
for (int i=0; ; i++)  
{  
    System.out.println(i);  
}
```



- Struktur einer HTML-Seite

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
  "http://www.w3.org/TR/html4/strict.dtd">
```

```
<html>
```

```
  <head><title>Info 1</title></head>
```

```
  <body><h1>Info 1 Webseite</h1><p>Hallo Welt!</p></body>
```

```
</html>
```

- Beschrieben durch DTD

```
<!ELEMENT html (head, body)>
```

```
<!ELEMENT head (title?)>
```

```
<!ELEMENT body (p|h1|br)*>
```

```
<!ELEMENT title (#PCDATA)>
```

```
<!ELEMENT p ANY>
```

```
<!ELEMENT h1 (#PCDATA)>
```

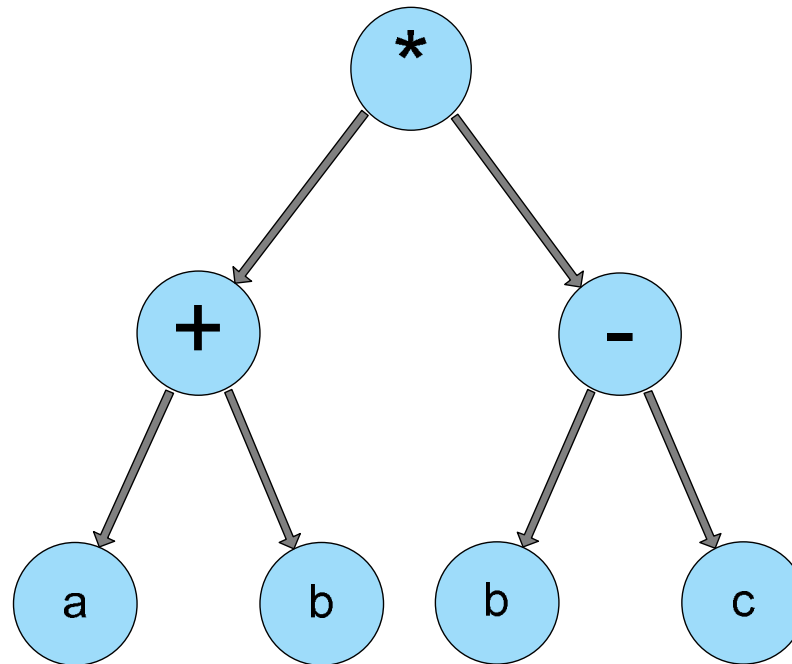
```
<!ELEMENT br EMPTY>
```



Aufgabe:

Zeichnen sie den Kantorowitschbaum folgender Formel:

$$(a + b) * (b - c)$$



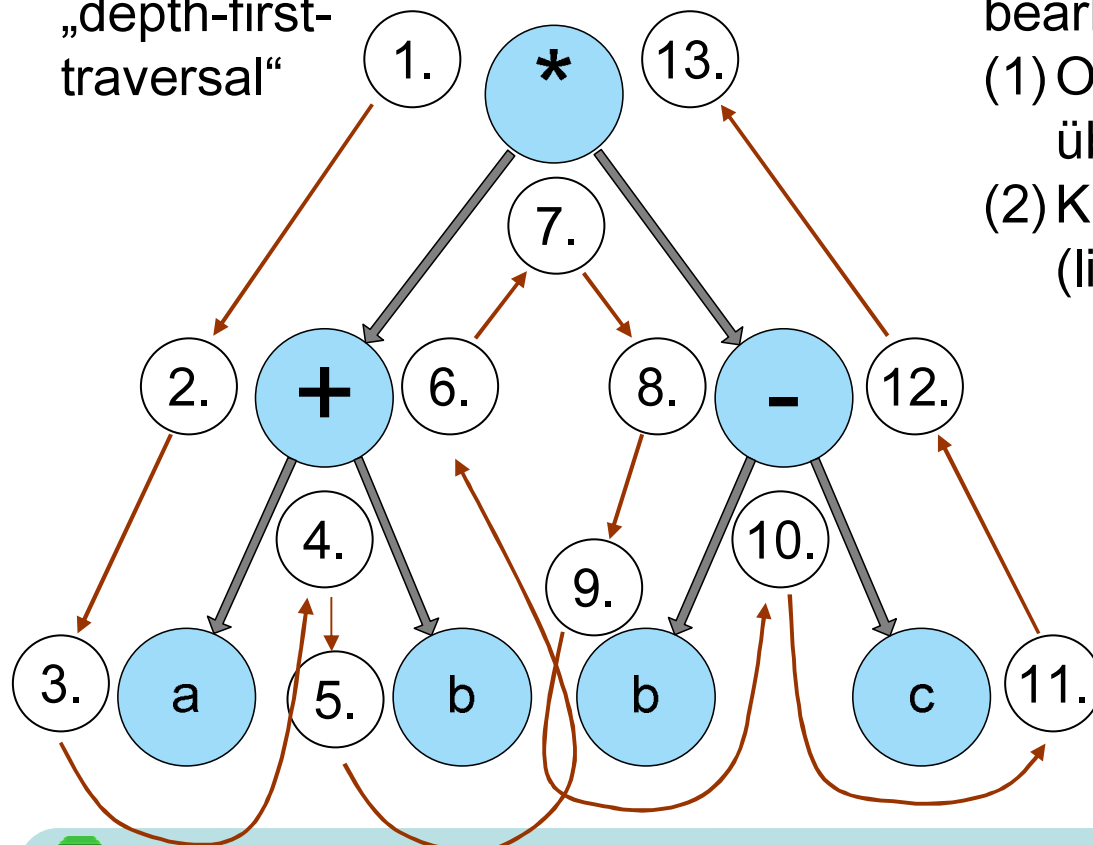
Aufgabe:

gegeben: Infixdarstellung:

$$(a + b) * (b - c)$$

Lesen Sie aus dem Baum die **Präfixschreibweise** der Formel ab:

„depth-first-traversal“



Algorithmus zum (Teil-)Bäume bearbeiten:

- (1) Operator (Knotenbeschriftung) übernehmen
- (2) Kinderteilbäume bearbeiten (linken zuerst, dann rechten)

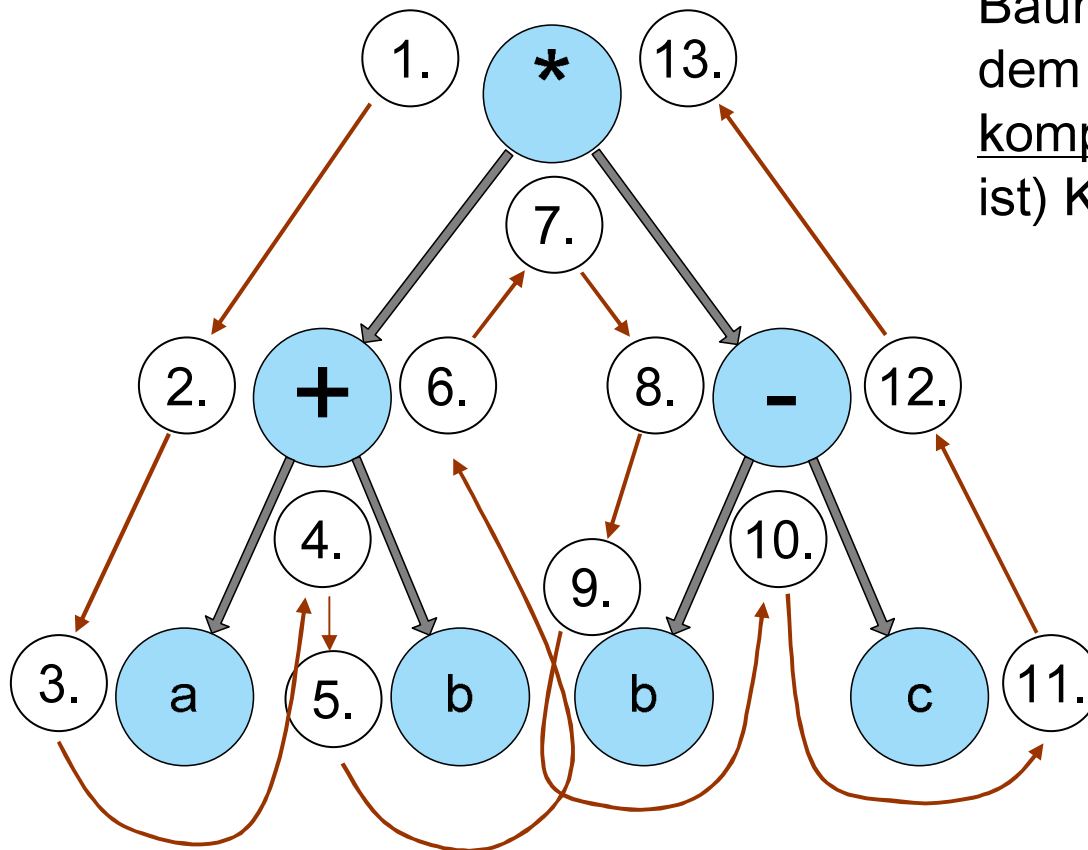
Ergebnis:

* + a b - b c



Lesen Sie aus dem Baum die **Postfixschreibweise** der Formel ab:

Algorithmus umgekehrt: erst in die Bäume hinabsteigen und auf dem „Rückweg“ (wenn der komplette Teilbaum bearbeitet ist) Knotenwert aufschreiben



Ergebnis:
 $a b + b c - *$

Zusammenfassung:

Infix: $(a + b) * (b - c)$

Präfix: $* + a b - b c$

Postfix: $a b + b c - *$



Aufgabe:

Gegeben sei die Grammatik $G = (\Sigma, \mathcal{N}, \mathcal{P}, S)$ mit $\mathcal{N} = \{S, B, X\}$,
 $\Sigma = \{a, b, c\}$ und

$$\mathcal{P} = \{ \begin{array}{l} S \mapsto aX \mid B \mid a \\ X \mapsto aX \mid B \\ B \mapsto bcB \mid bc \end{array} \}$$

(2) Die Kettenproduktionen lassen sich auch in nur lineare bzw. terminierende Produktionen auflösen, z.B.:
 $S \mapsto aX \mid bB' \mid bB'' \mid a$

Geben Sie zum Ausdruck aabcbc den Ableitungsbaum an.

(1) Diese Regel ist keine lineare Produktion nach unserer Definition. Offensichtlich lassen sich jedoch leicht Produktionen angeben, die die gleichen Teilworte erzeugen, jedoch linear bzw. terminierend sind:

$$\begin{array}{ll} B & \mapsto bB' \mid bB'' \\ B' & \mapsto cB \\ B'' & \mapsto c \end{array}$$

Die erzeugte Sprache ist also regulär ($\mathcal{L}(G)$ ist in der Klasse der CH-3 Sprachen)

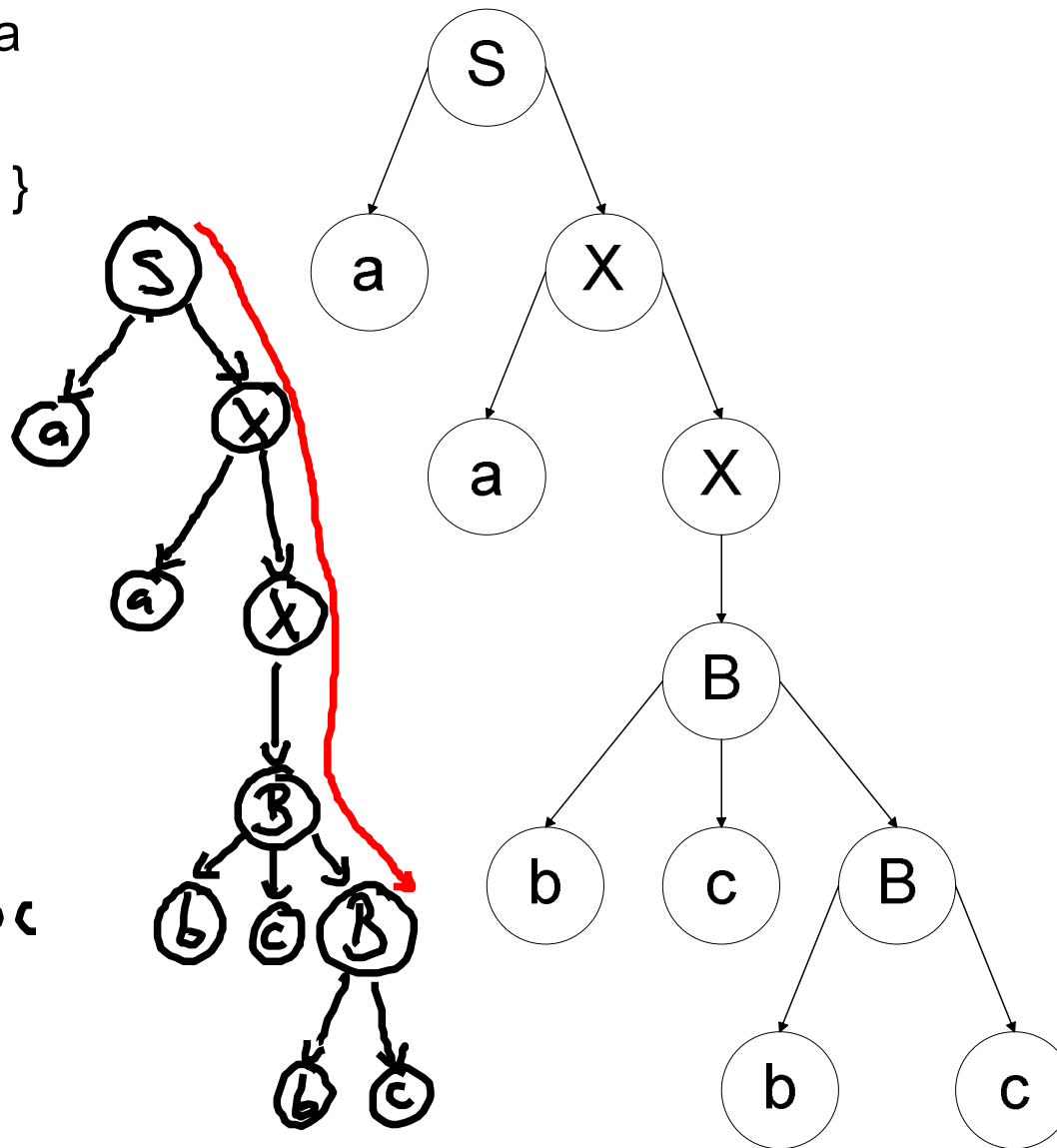


Ableitungsbaum

$\mathcal{P} = \{$
 $S \mapsto aX \mid B \mid a$
 $X \mapsto aX \mid B$
 $B \mapsto bcB \mid bc \}$

Ausdruck: aabcabc

$S \Rightarrow aX$
 $aX \Rightarrow aaX$
 $aaX \Rightarrow aaB$
 $aaB \Rightarrow aabcB$
 $aabcB \Rightarrow aabcabc$



Backus-Naur-Form (kurz BNF)

- statt \rightarrow schreibe $::=$
- Nichtterminale in $\langle \rangle$
- senkrechter Strich trennt Alternativen |

'''
'''
'''
'''

erweiterte Backus-Naur-Form (kurz EBNF)

- optionale Teile in [...]
- Gruppe von Zeichen, z.B. Phrase in (...)
- alternative Klammerinhalte durch | getrennt
- * nach Zeichen oder Gruppe gibt beliebig viele Wiederholungen an (**auch gar keine**)
- + nach Zeichen oder Gruppe gibt beliebig viele Wiederholungen an (**aber: mindestens eine**)
- Terminale in Anführungszeichen '...' (Unterscheidung von Metazeichen)

[...]⁺ \equiv ()^{*}



Aufgabe:

Beschreiben Sie mit BNF Regeln die Dezimalschreibweise; dabei sind nur Zahlen zugelassen, die keine führenden Nullen und keine Null hinter der letzten von Null verschiedenen Nachkommastelle besitzen.

$\langle \text{Dezimalzahl} \rangle ::= \langle \text{Vorzeichen} \rangle \langle \text{Vorkommazahl} \rangle , \langle \text{Nachkommazahl} \rangle$

$\langle \text{Vorzeichen} \rangle ::= + \mid -$

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

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

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

$\langle \text{Nichtnull} \rangle ::= 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{Ziffer} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$



Aufgabe:

Wandeln Sie nun die BNF in eine EBNF um.

```
<Dezimalzahl> ::= <Vorzeichen> <Vorkommazahl> , <Nachkommazahl>  
<Vorzeichen> ::= + | -  
<Vorkommazahl> ::= <Ziffer> | <Nichtnull> <Zahl>  
<Nachkommazahl> ::= <Ziffer> | <Zahl> <Nichtnull>  
<Zahl> ::= <Ziffer> | <Ziffer> <Zahl>  
<Nichtnull> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9      (BNF)
```

```
<Dezimalzahl> ::= [ '+' | '-' ] <Vorkommazahl> , <Nachkommazahl>  
<Vorkommazahl> ::= 0 | <Nichtnull> <Ziffer> *  
<Nachkommazahl> ::= <Ziffer> * <Nichtnull>  
<Nichtnull> ::= '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'  
<Ziffer> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'      (EBNF)
```



Halbgruppe:

- Menge \mathcal{U} mit binärer Operation \cdot , so dass für alle $a, b, c \in \mathcal{U}$ gilt:
 - Abgeschlossenheit: $a \cdot b \in \mathcal{U}$
 - Assoziativgesetz: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$

Monoid:

- Halbgruppe (\mathcal{U}, \cdot) mit Element $\varepsilon \in \mathcal{U}$, so dass für alle $a \in \mathcal{U}$ gilt:
 - Einselement: $\varepsilon \cdot a = a$ und $a \cdot \varepsilon = a$

Ein Monoid bzw. Halbgruppe heißt abelsch, wenn für alle $a \in \mathcal{U}$ gilt:

- Kommutativgesetz: $a \cdot b = b \cdot a$



Beispiel 1:

- Struktur $(+, 2\mathbb{Z})$ mit normaler Addition auf den geraden ganzen Zahlen

Abgeschlossenheit: $2k + 2m = 2(k+m) \in 2\mathbb{Z} \quad \checkmark$

Assoziativität: $2k + (2m + 2p) = 2(k+m+p) = (2k+2m) + 2p \quad \checkmark$

Einselement: $2k + e = 2k \Rightarrow e = 0 \in 2\mathbb{Z} \quad \checkmark$

$e + 2k = 2k \Rightarrow e = 0 \quad \checkmark$

kommutativ: $2k + 2m = 2(k+m) = 2(m+k) = 2m + 2k \quad \checkmark$

- Struktur $(+, 2\mathbb{Z}+1)$ mit Addition auf den ungeraden ganzen Zahlen

Abgeschlossenheit: $2k+1 + 2m+1 = 2(k+m+1) \in 2\mathbb{Z}$



Beispiel 1:

- Struktur $(+, 2\mathbb{Z})$ mit normaler Addition auf den geraden ganzen Zahlen

Seien $e, k, m, p \in \mathbb{Z}$

Abgeschlossenheit: $2k + 2m = 2(k+m) \in 2\mathbb{Z}$

Assoziativität: $2k + (2m + 2p) = 2(k + m + p) = (2k + 2m) + 2p$

Einselement: $2k + e = 2k \Rightarrow e = 0 \in 2\mathbb{Z}$

$$e + 2k = 2k \Rightarrow e = 0 \in 2\mathbb{Z}$$

Kommutativität: $2k + 2m = 2(k+m) = 2(m+k) = 2m + 2k$

Die Struktur ist also ein abelscher Monoid.

- Struktur $(+, 2\mathbb{Z}+1)$ mit Addition auf den ungeraden ganzen Zahlen

Abgeschlossenheit: $2k+1 + 2m+1 = 2(k+m+1) \in 2\mathbb{Z}$

Für diese Verknüpfung ist also die Abgeschlossenheit nicht gegeben,
z.B. $1 + 1 = 2$!



Beispiel 2:

- Struktur (\diamond, \mathbb{R}) mit Verknüpfung \diamond auf den reellen Zahlen definiert durch:

$$a \diamond b = a + (k \cdot b), \quad k \in \mathbb{R} \text{ fest}$$

Abgeschlossenheit: ✓

Assoziativität: $a \diamond (b \diamond c) = a \diamond (b + k \cdot c) = a + kb + k^2c$
 $\neq a + kb + kc = (a + kb) \diamond c = (a \diamond b) \diamond c$

- Wähle nun k so, dass (\diamond, \mathbb{R}) zu einer Halbgruppe oder Monoid wird:

$$k := 1 \Rightarrow a \diamond b = a + b$$



Beispiel 2:

- Struktur (\diamond, \mathbb{R}) mit Verknüpfung \diamond auf den reellen Zahlen definiert durch:

$$a \diamond b = a + (k \cdot b), k \in \mathbb{R} \text{ fest}$$

Abgeschlossenheit: klar

Assoziativität: $a \diamond (b \diamond c) = a \diamond (b + kc) = a + kb + k^2c$
 $(a \diamond b) \diamond c = (a + kb) \diamond c = a + kb + kc$
 \Rightarrow nicht assoziativ
 \Rightarrow keine Halbgruppe

- Wähle nun k so, dass (\diamond, \mathbb{R}) zu einer Halbgruppe oder Monoid wird:
Wähle $k = 1 \Rightarrow a \diamond b = a + b$

Dies ist normale Addition auf reellen Zahlen. Sie ist assoziativ, kommutativ, abgeschlossen und hat die Null als Einselement
 \Rightarrow Die Verknüpfung ist ein abelscher Monoid



[Tacker]

Matrikelnummer

Übungsblatt X

Tut.nr.

Name, Vorname

Informatik I

Bla blabla blubber rababer gelaber. Blablabla blub bla. Blablabla! Bluber blub blubbla. Blabberblubber laber. Rababer blaber blub laber rababer. Bla blub Rababera baberaba bla. Blubber bla rababer gelaber. Blubber blubber blubber blubber blubber blubber. Bla blabla blubber rababer gelaber. Blablabla blub bla. Blablabla! Bluber blub blubbla. Blabberblubber blubber blubber blubber blubber blubber. Bla blabla blubber rababer gelaber. Blablabla blub bla. Blablabla! Bluber blub blubbla. Blabberblubber laber. Rababer blaber blub laber rababer. Bla blub Rababera baberaba bla. Blubber bla! Bluber blub blubbla. Blabberblubber laber. Rababer blaber blub laber rababer. Bla blub Rababera baberaba bla. Blubber bla rababer gelaber. Blubber blubber blubber blubber blubber blubber....



- Sowie die Musterlösungen finden Sie unter

<http://www.infoeins.de/uebungsblaetter.php>

