



Übungsblatt 2

- Grammatiken und logische Ausdrücke -

Aufgabe 1: Formale Sprachen: Grammatiken

Gegeben sei eine Grammatik $G = \langle \Sigma, N, P, S \rangle$ mit:

Mit dem Alphabet: $\Sigma = \{a, b\}$

Den Nichtterminalsymbolen: $N = \{S, A, B\}$

Dem Startsymbol: $S = S$

und den Produktionsregeln:

$$P = \begin{cases} S \rightarrow aA \mid bB \\ A \rightarrow bA \mid a \\ B \rightarrow aB \mid b \end{cases}$$

1. Bestimmen Sie drei Wörter, die von dieser Grammatik erzeugt werden können.

aa, bb, abba, baab, ...

2. Geben Sie die Ableitungsschritte für eines dieser Wörter an.

$S \vdash aA \vdash abA \vdash abba$

3. Beschreiben Sie mit eigenen Worten, welche Sprache $L(G)$ diese Grammatik erzeugt (z. B. Alle Wörter starten mit ...).

$L(G) = \{w|ab^na, n \in \mathbb{N}_0\} \cup \{w|ba^nb, n \in \mathbb{N}_0\}$

Die Wörter starten und enden entweder mit einem a oder einem b . Wenn sie mit einem a starten, können sie beliebig viele b 's enthalten. Wenn Sie mit einem b starten können Sie beliebig viele a 's enthalten.

4. Ist die Grammatik vom Typ 2 oder Typ 3? Begründen Sie Ihre Antwort.

Es handelt sich um eine rechtlineare Typ 3 Grammatik. Alle Produktionsregel haben die Form:

$P \subseteq N \times (N\Sigma \cup \Sigma \cup \{\varepsilon\})$

Aufgabe 2: Hexadezimalkonstanten in Java

Schauen Sie sich die Grammatikregeln zu Hexadezimalkonstanten in Java an (Nichtterminalsymbol $\langle \text{HexNumeral} \rangle$). Sie finden die Grammatikregeln dazu in Kapitel 3.10.1 des [Javasprachstandards](#). Hexadezimalkonstanten sind eine andere Darstellungsmöglichkeit für Zahlen, die wir später noch intensiver betrachten werden. Geben Sie mit eigenen Worten an, was diese Definitionen insgesamt beschreiben / welche Kombinationen von Zeichenfolgen damit erzeugt werden können. Sie sollen also nicht die einzelnen Regeln erläutern, sondern vielmehr die durch die Gesamtheit dieser Regeln erzeugbaren Wörter kompakt beschreiben.

Geben Sie Ableitungen an von $\langle \text{HexNumeral} \rangle$:

1. zu einem möglichst kurzen Wort nur aus Terminalsymbolen

$\langle \text{HexNumeral} \rangle \vdash 0x \langle \text{HexDigits} \rangle \vdash 0x \langle \text{HexDigit} \rangle \vdash 0xF$

2. zu einem Wort nur aus Terminalsymbolen, in dessen Ableitung alle Regeln angewandt werden

$\langle \text{HexNumeral} \rangle$

$\vdash 0x \langle \text{HexDigits} \rangle$

$\vdash 0x \text{ <HexDigit> <HexDigitsAndUnderscores> <HexDigit>}$
 $\vdash 0x0 \text{ <HexDigitsAndUnderscores> <HexDigit>}$
 $\vdash 0x0 \text{ <HexDigitOrUnderscore> <HexDigit>}$
 $\vdash 0x0_ \text{ <HexDigit>}$
 $\vdash 0x0_0$

Aufgabe 3: Algorithmische Grundbausteine

1. Erstellung einer eigenen Grammatik

In der Vorlesung wurden algorithmische Grundbausteine (Einzelanweisung, Sequenz, Selektion, Mehrfachselektion, Iteration) vorgestellt, die alle auf dem Begriff einer Anweisung aufbauen. Geben Sie eine (vollständige) Grammatik in EBNF für Anweisungen an, die alle vorgestellten Grundbausteine abdeckt. Das Startsymbol der Grammatik ist <Anweisung> und daraus müssen sich alle Bausteine erzeugen lassen (und nur die). Lassen Sie die rechte Seite der Grammatikregel zu einer Einzelanweisung frei, dies wollen wir hier nicht weiter spezifizieren. In einer Selektion benötigen Sie zum Beispiel auch eine *Bedingung*. Lassen Sie in so einem Fall ebenso die rechte Seite einer Regel für Bedingung frei, also nicht weiter spezifiziert. Überlegen Sie, was Terminalsymbole dieser Grammatik sind und welche Nichtterminalsymbole Sie benötigen (dies ergibt sich über ihre Regeln).

Gegeben sei eine Grammatik $G = \langle \Sigma, N, P, S \rangle$ mit:

Mit dem Alphabet: $\Sigma = \{\text{if, then, else, switch, case, while, do, :, ;}\}$

Den Nichtterminalsymbolen: $N = \{\text{<Anweisung>, <Sequenz>, <Selektion>...}\}$

Dem Startsymbol: $S = \text{<Anweisung>}$

und den Produktionsregeln: $P = \{$

< Anweisung >::=	$\text{< Einzelanweisung > < Sequenz > < Selektion > < Mehrfachselektion > < Iteration >}$
< Sequenz >::=	$\text{< Anweisung > \{ < Anweisung > \}}$
< Selektion >::=	$\text{if < Bedingung > then < Anweisung > [else < Anweisung >]}$
< Iteration >::=	$\text{while < Bedingung > do < Anweisung >}$
$\text{< Mehrfachselektion >::=}$	$\text{switch < Ausdruck > \{ < caseAnweisung > \}}$
$\text{< caseAnweisung >::=}$	$\text{case < Konstante > : < Anweisung >;}$
$\text{< Einzelanweisung >::=}$	\dots
< Bedingung >::=	\dots
< Ausdruck >::=	\dots
< Konstante >::=	\dots

$\}$

2. Ableitung des Euklidischen Algorithmus

Geben Sie zu Ihren Grammatikregeln eine Ableitung an für das Programm auf Folie 22 (*02_algorithmen*), wo ja bereits die Struktur des Programms farblich gekennzeichnet ist.

<Anweisung>
 $\vdash \text{<Selektion>}$
 $\vdash \text{if <Bedingung> then <Anweisung>}$
 $\vdash \text{if <Bedingung> then <Sequenz>}$
 $\vdash \text{if <Bedingung> then <Anweisung> <Sequenz>}$
 $\vdash \text{if <Bedingung> then <Anweisung> <Anweisung>}$
 $\vdash \text{if <Bedingung> then <Iteration> <Anweisung>}$
 $\vdash \text{if <Bedingung> then <Iteration> <Einzelanweisung>}$
 $\vdash \text{if <Bedingung> then while <Bedingung> do <Anweisung> <Einzelanweisung>}$
 $\vdash \text{if <Bedingung> then while <Bedingung> do <Selektion> <Einzelanweisung>}$
 $\vdash \text{if <Bedingung> then while <Bedingung> do}$

```
if <Bedingung> then <Anweisung> else <Anweisung> <Einzelanweisung>
└ if <Bedingung> then while <Bedingung> do if <Bedingung> then <Einzelanweisung> else
  <Einzelanweisung> <Einzelanweisung>
```

Aufgabe 4: Gesetzmäßigkeiten für die logischen Operatoren

Formulieren Sie für die Java-Operationen & (Logisches Und) und | (Logisches Oder) das Kommutativgesetz, das Assoziativgesetz und das Distributivgesetz. Wie lauten die De Morgan'schen Regeln?

Kommutativgesetz:

$x \& y == y \& x$

$x | y == y | x$

Assoziativgesetz:

$(x \& y) \& z == x \& (y \& z)$

$(x | y) | z == x | (y | z)$

Distributivgesetz:

$(x \& y) | z == (x | z) \& (y | z)$

$(x | y) \& z == (x \& z) | (y \& z)$

de Morgansche Regeln:

$!(x \& y) == !x | !y$

$!(x | y) == !x \& !y$

Aufgabe 5: Vereinfachung von Ausdrücken

Vereinfachen Sie folgende logische Ausdrücke:

1. $(x < 100) | !(x < 200)$

$(x < 100) | (x \geq 200)$

2. $x \& !(y \& !(z | y))$

Nutzen Sie ausschließlich Java-Syntax für logische Ausdrücke, wie dies in den vorgegeben Ausdrücken auch schon geschehen ist. Eine Umformung eines Ausdrucks in einen anderen äquivalenten Ausdruck können Sie durch == kennzeichnen.

Beispiel dazu: $a \& b == b \& a$.

```
x & !(y & !(z | y))
= x & !(y & (!z & !y))
= x & !(y & !z & !y)
= x & !(y & !y & !z)
= x & !(false & !z)
= x & (true | z)
= x & true
= x
```

Aufgabe 6: Gültige Wertkombinationen

Für welche *ganzzahligen* Wertkombinationen der vorkommenden Variablen sind folgenden logische Ausdrücke wahr?

1. $(x < y) \& (y < z) \& (x > 100) \& (z < 200)$

$x \in [101, 197], y \in [102, 198], z \in [103, 199], x < y < z$

2. $(x > -2) \& (x < 8) \& (x \text{ ist gerade})$

$x \in \{0, 2, 4, 6\}$

Aufgabe 7: Monate mit 30 und 31 Tagen

Geben Sie einen logischen Ausdruck an, der anhand einer Monatsangabe (Januar=1, Februar=2, . . . Dezember=12) feststellt, ob der Monat 31 Tage hat.

```
((monat <= 7) & (monat ungerade)) | ((monat > 7) & (monat gerade))
```

Aufgabe 8: Berechnung von Schaltjahren

Geben Sie einen logischen Ausdruck an, der anhand einer Jahresangabe $j > 0$ feststellt, ob das Jahr ein Schaltjahr ist/war. Beachten Sie dabei den Unterschied zwischen Julianischem und Gregorianischem Kalender ab 1582.

```
(j durch 4 teilbar) & ((j <= 1582) | !(j durch 100 teilbar) | (j durch 400 teilbar))
```

Aufgabe 9: Auswertung logischer Ausdrücke

Rechnen Sie zuerst auf dem Papier folgende logische Ausdrücke per Hand aus und implementieren diese anschließend in Java. wert1 und wert2 seien logische Variablen mit den Wahrheitswerten wert1=true beziehungsweise wert2=false.

1. (wert1 & wert2) | wert1

```
(true & false) | true = false | true = true
```

2. (wert1 & wert2) & !wert1

```
(true & false) & false = false & false = false
```

3. !((!wert1) & wert2)

```
!((!true) & false) = !(false & false) = !false = true
```

4. !(((5 < 7) & wert1) || wert2)

```
!(((5 < 7) & true) || false) = !((true & true) || false) = !(true || false) = !true = false
```

Aufgabe 10: 🐞 Bestimmung von logischen Ausdrücken

Schreiben Sie ein Java-Programm mit dem Namen BoolExpression, das zwei Variablen wert1 und wert2 des Typs boolean enthält. wert1 soll zu Beginn den Wert false enthalten, wert2 soll den Wert true enthalten. Berechnen Sie in Java dann das Ergebnis des logischen Ausdrucks !wert1 & (wert1 | wert2) und geben Sie das Resultat dieses Ausdrucks auf dem Bildschirm aus. Die Ausgabe dazu wird also aus genau einer Zeile bestehen, in der entweder true oder false steht (**und sonst nichts**). Ändern Sie anschließend im Programm die Variable wert1 so, dass sie den Wert true enthält. Berechnen Sie den obigen Ausdruck erneut in ihrem Programm und geben wiederum den Ergebniswert des Ausdrucks wie oben auf dem Bildschirm aus.

Die Ausgabe des Programms besteht also aus genau zwei Zeilen, die jeweils entweder true oder false enthalten.

Info

- Sie können das Ergebnis jeweils entweder in einer weiteren Variablen zwischenspeichern oder den o.g. logischen Ausdruck direkt in `System.out.println()` verwenden.
- Sie sollen ein Programm entwickeln, das nacheinander die Variablen deklariert, den Ausdruck auswertet und das Resultat ausgibt, dann den Wert der Variablen wert1 verändert und dann diesen Ausdruck auswertet und das Ergebnis dazu ausgibt.

Aufgabe 11: Von Zahlen zu logischen Ausdrücken

Geben Sie Java-Formulierungen für folgende logische Bedingungen an (i,j, und k stehen für int-Variablen beliebigen Inhalts)

1. i, j und k sind alle echt kleiner als 50

```
(i < 50) && (j < 50) && (k < 50)
```

2. die Summe von i,j und k ist nicht durch 7 teilbar

```
((i + j + k) % 7) != 0
```

3. j ist ungerade und ist entweder im Intervall [3, 21] oder im Intervall [112, 157]

```
(j % 2 == 1) && (((j >= 3) && (j <= 21)) || ((j >= 112) && (j <= 157)))
```

4. k ist weder durch 3 noch durch 7 teilbar, aber gerade

```
(k % 3 != 0) && (k % 7 != 0) && (k % 2 == 0)
```

Info

Bei [112, 157] handelt es sich um ein abgeschlossenes Intervall ($112 \leq x \leq 157$)