Prof. Dr. Stephan Kleuker
Hochschule Osnabrück
Fakultät Ing.-Wissenschaften und Informatik
- Software-Entwicklung -

**Theoretische Informatik**

Sommersemester 2023
**5. Aufgabenblatt**

**Aufgabe 17 (Transformation kontextfreier Grammatiken [klausurähnlich], 3+3+2 Punkte)**
Gegeben sei jeweils folgende Grammatik ({Start, A, B}, {a, b}, Regeln, Start) mit den Regeln

a) Start -> aAB  A -> ABBA | B  B -> b | ε
   Geben Sie eine sprachäquivalente Grammatik (Regelmenge) ohne ε-Regeln an.
b) Start -> AB  A -> ABBA | B  B -> b | A
   Geben Sie eine sprachäquivalente Grammatik (Regelmenge) ohne Ketten-Regeln an.
c) Start -> AB  A -> ABBA | a  B -> b | AAA
   Geben Sie eine sprachäquivalente Grammatik (Regelmenge) in Chomsky-Normalform an.

**Aufgabe 18 (Transformation kontextfreier Grammatiken [klausurähnlich], 3+3+2 Punkte)**
Gegeben sei jeweils folgende Grammatik ({Start, A, B}, {a, b}, Regeln, Start) mit den Regeln

a) Start -> ABC  A -> AAB | ε  B -> b | ε  C -> CC | c
   Geben Sie eine sprachäquivalente Grammatik (Regelmenge) ohne ε-Regeln an.
b) Start -> ABC  A -> BB  B -> b | A  C -> c | A
   Geben Sie eine sprachäquivalente Grammatik (Regelmenge) ohne Ketten-Regeln an.
c) Start -> ABC  A -> ABC | a  B -> b | ABC  C -> ABC | c
   Geben Sie eine sprachäquivalente Grammatik (Regelmenge) in Chomsky-Normalform an.

**Aufgabe 19 (Grammatiktransformationen)**

a) Geben Sie Ihre Lösung aus 17 a) in die Datei KFGOhneEps.kfg ein und prüfen Sie mit OhneEpsTest.java ob Ihre Lösung richtig sein könnte.
b) Geben Sie Ihre Lösung aus 17 b) in die Datei KFGOhneKette.kfg ein und prüfen Sie mit OhneKetteTest.java ob Ihre Lösung richtig sein könnte.
c) Geben Sie Ihre Lösung aus 17 c) in die Datei KFGInChomsky.kfg ein und prüfen Sie mit InChomskyTest.java ob Ihre Lösung richtig sein könnte.
d) Geben Sie Ihre Lösung aus 18 a) in die Datei KFGOhneEps2.kfg ein und prüfen Sie mit OhneEps2Test.java ob Ihre Lösung richtig sein könnte.
e) Geben Sie Ihre Lösung aus 18 b) in die Datei KFGOhneKette2.kfg ein und prüfen Sie mit OhneKette2Test.java ob Ihre Lösung richtig sein könnte.
f) Geben Sie Ihre Lösung aus 18 c) in die Datei KFGInChomsky2.kfg ein und prüfen Sie mit InChomsky2Test.java ob Ihre Lösung richtig sein könnte.
g) Formen Sie folgende Grammatiken schrittweise in die Chomsky-Normalform um. Schreiben Sie Ihr Ergebnis in die angegebene Datei und nutzen Sie die angegebene Testklasse um zu prüfen, ob Ihr Ergebnis richtig sein kann.

| | Grammatik, Start ist X | Datei | Test |
|---|---|---|---|
| I. | X -> A \| B<br>A -> AAA \| aBBB<br>B -> b \| ε | KFGTrafo1.kfg | KFGTrafo1Test.java |
| II. | X -> aXa \| A<br>A -> bAb \| B<br>B -> ε | KFGTrafo2.kfg | KFGTrafo2Test.java |
| III. | X -> aaX \| A<br>A -> bbA \| B<br>B -> ccX \| ε | KFGTrafo3.kfg | KFGTrafo3Test.java |

**Aufgabe 20 (programmatische Konstruktion kontextfreier Grammatiken; Ansatz wird auf Folgeblättern genutzt)**
Kontextfreie Grammatiken können in der gegebenen Bibliothek auch rein aus Objekten schrittweise konstruiert werden. Dazu stehen einige Informationen im Anhang.

a) Schreiben Sie eine Grammatik für die Sprache $\{a^n b^{n+2} \mid n \geq 0\}$ ausschließlich aus Objekten bestehend in Java, also ohne Textdatei oder String-Variable mit der Grammatik als Text in einem Beispielprogramm und lassen Sie sich einen zu Ihrer Grammatik gehörenden Ableitungsbaum für das Wort aabbbb als Graphik anzeigen, um dann eine JPG-Datei zu generieren.
b) Die in der Vorlesung vorgestellte Grammatik zu einer Programmiersprache steht in der Datei „Programmiersprache.kfg“. Schreiben Sie ein Programm in dieser Sprache mit dem die Werte in den Variablen x1 und x2 (vom Typ 2) vertauscht werden. Zeigen Sie mit der Java-Bibliothek, dass Ihr Programm syntaktisch korrekt ist und lassen Sie sich ein JPG mit einem Ableitungsbaum generieren. Die Grammatik können Sie natürlich aus der Datei einlesen.

**Aufgabe 21 ( eigene Umsetzung von Transformationsalgorithmen; aufwändig)**
In der gegebenen Bibliothek sind bereits die Transformationsalgorithmen aus der Vorlesung umgesetzt. Dabei wurden die Algorithmen nur prototypisch ohne jedwede Optimierung programmiert. Lassen Sie eine Klasse von KontextreieGrammatik.java erben und überschreiben Sie die folgenden oder einen der folgenden Algorithmen.

`public boolean ohneLeeresWortRegeln():` transformiert die Grammatik sprachäquivalent so, dass es keine Regeln mit dem leeren Wort auf der rechten Seite gibt; das Ergebnis gibt an, ob das leere Wort von der Grammatik erzeugt werden kann
`public KontextfreieGrammatik entferneKettenregeln():` transformiert die Grammatik sprachäquivalent so, dass es keine Kettenregeln mehr gibt; das Ergebnis ist this
`public KontextfreieGrammatik inChomskyNormalform():` transformiert die Grammatik sprachäquivalent in eine Grammatik in Chomsky-Normalform; das Ergebnis ist this

Tragen Sie Ihre Klasse in test.kontextfreieGrammatik.OptimierungsTest.java in Zeile 22 ein und lassen Sie das Programm laufen. Es wird versucht, die Korrektheit zu prüfen und an Beispielen grob die Performance analysiert.
Sie können zur Bearbeitung auch die Klasse grammatik.OptimierteGrammatik.java nutzen, die einfach eine Kopie der existierenden Algorithmen enthält, die überarbeitet oder völlig neu konzipiert werden können. Anregungen für ein neues Konzept können z. B. dem Skript von Karsten Morisse entnommen werden, da es dort meist leicht abgewandelte Ansätze gibt.
Schwankungen bis mindestens 3% können durch die Laufzeitumgebung auftreten.

Kontextfreie Grammatiken als Objekte
Das nachfolgende Programm zeigt Möglichkeiten Grammatiken aus Java-Objekten verschiedener Domänen-Klassen zusammenzusetzen. Der Ansatz ist besonders dann interessant, falls die Grammatik weiterverarbeitet werden soll, wenn ihr z.B. eine Semantik zugordnet wird. Bei der Methode ignoriereLeerzeichen(true) werden Leerzeichen aus Worten entfernt, deren Syntax überprüft werden soll. Alternativ kann new Wort(String) genutzt werden, wodurch Leerzeichen erhalten bleiben und dann natürlich auch als Terminalzeichen /space existieren sollten. Die umgesetzte Grammatik ist: Nichtterminale = {X, A, B}, Terminale = {a,b,c}, Startsymbol ist X und die Regeln sind:

X -> aaX | A
A -> bbA | B
B -> ccX | ε

```
package main;

import alphabet.Nichtterminal;
import alphabet.Terminal;
```

```java
import ausfuehrung.Ableitung;
import grammatik.KontextfreieGrammatik;
import grammatik.KontextfreieRegel;
import semantik.Ableitungsbaum;

public class KontextfreieGrammatikProgrammiert {

  public static void main(String[] args) {
    KontextfreieGrammatik g = new KontextfreieGrammatik();
    g.setIgnoriereLeerzeichen(true);
    Nichtterminal nX = Nichtterminal.nichtterminal("X");
    Nichtterminal nA = Nichtterminal.nichtterminal("A");
    Nichtterminal nB = Nichtterminal.nichtterminal("B");
    g.addNichtterminale(nX, nA, nB);
    Terminal ta = Terminal.terminal("a");
    Terminal tb = Terminal.terminal("b");
    Terminal tc = Terminal.terminal("c");
    g.addTerminale(ta, tb, tc);
    g.setStart(nX);
    KontextfreieRegel r1 = new KontextfreieRegel(nX, g.stringAlsWort("aaX"));
    KontextfreieRegel r2 = new KontextfreieRegel(nX, g.stringAlsWort("A"));
    KontextfreieRegel r3 = new KontextfreieRegel(nA, g.stringAlsWort("bbA"));
    KontextfreieRegel r4 = new KontextfreieRegel(nA, g.stringAlsWort("B"));
    KontextfreieRegel r5 = new KontextfreieRegel(nB, g.stringAlsWort("ccX"));
    KontextfreieRegel r6 = new KontextfreieRegel(nB, g.stringAlsWort("/eps"));
    g.addRegeln(r1, r2, r3, r4, r5, r6);
    System.out.println(g);

    Ableitung abl = g.ableitbar(g.stringAlsWort("aabbcc"));
    System.out.println("Ist aabbcc ableitbar: " + abl.isErfolgreich());
    Ableitungsbaum ablb = Ableitungsbaum.ableitungAlsBaum(abl);
    ablb.linksdurchlauf();
    ablb.visualisieren();

    Ableitung abl2 = g.berechneAbleitung(g.stringAlsWort("aabbcc"));
    System.out.println("\nIst aabbcc ableitbar: " + abl2.isErfolgreich());
    Ableitungsbaum ablb2 = Ableitungsbaum.ableitungAlsBaum(abl2);
    ablb2.linksdurchlauf();
    ablb2.visualisieren();
  }
}
```

Die zugehörige Ausgabe sieht wie folgt aus.

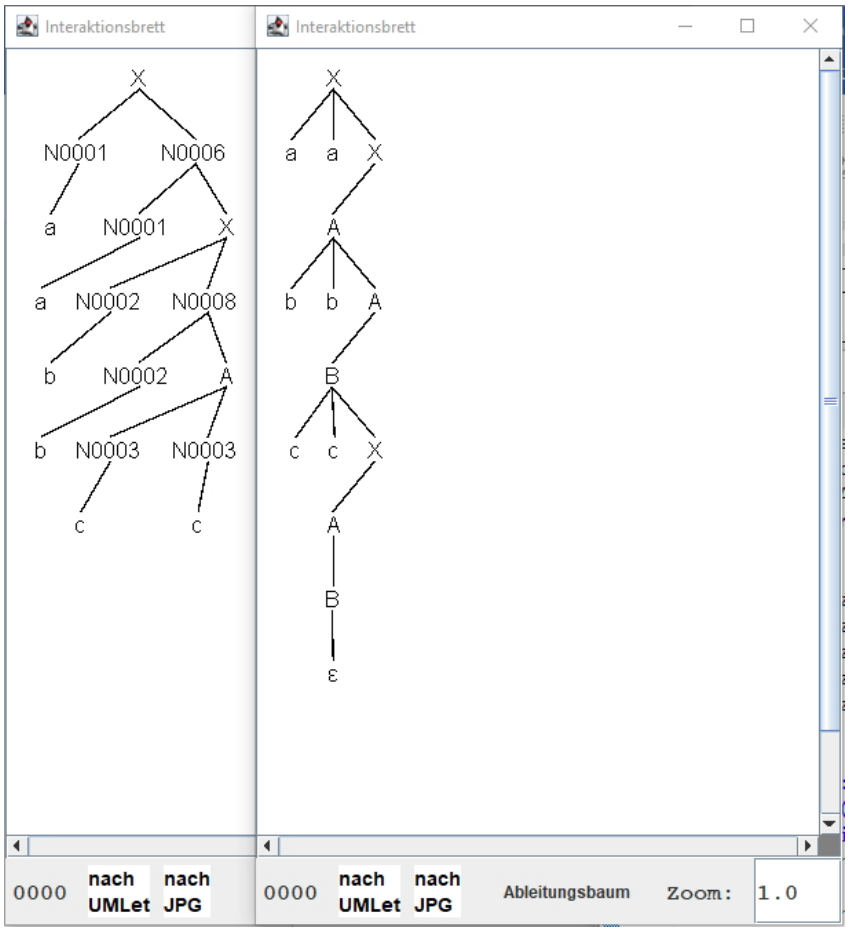
```
Terminale =[a, b, c]
Nichtterminale = [A, B, X]
Start = X
 (1) X -> aaX
 (2) X -> A
 (3) A -> bbA
 (4) A -> B
 (5) B -> ccX
 (6) B -> ε

Ist aabbcc ableitbar: true
0: X -> N0001N0006
```

```
1: N0001 -> a
2: N0006 -> N0001X
3: N0001 -> a
4: X -> N0002N0008
5: N0002 -> b
6: N0008 -> N0002A
7: N0002 -> b
8: A -> N0003N0003
9: N0003 -> c
10: N0003 -> c

Ist aabbcc ableitbar: true
0: X -> aaX
1: X -> A
2: A -> bbA
3: A -> B
4: B -> ccX
5: X -> A
6: A -> B
7: B -> ε
```

Die nachfolgende Abbildung zeigt die generierten graphischen Ausgaben. Es wird deutlich, dass bei der Nutzung von `ableitbar(.)` ein Ableitungsobjekt erzeugt wird, das zur Grammatik in Chomsky-Normalform passt. Wird die aufwändigere Methode `berechneAbleitung()` genutzt, bezieht sich die berechnete Ableitung auf die Ursprungsgrammatik. Das Ergebnis ist auf der rechten Seite dargestellt.



Unter den Ausgaben befinden sich zwei Knöpfe, mit denen das Ergebnis in eine JPG-Datei und in eine Datei des graphischen Werkzeugs UMLet verwandelt werden kann. Letzte ist zwar bearbeitbar, da aber keine pixelgenaue Platzierung unterstützt wird, wohl von eher geringem Nutzen.

Besteht ein reines Interesse an einer nicht-graphischen Ableitung ist diese mit der Methode `zeigeLinksableitung()` berechenbar, wie folgendes Beispiel zeigt.

```
public class KFGAbleitungAngebenLassen {

  public static void main(String[] args) {
    String grammatik =
        """
        N: S T
        T: a b
        S: S
        r0:: S -> TT
        r1:: T -> aTb | ab
        """;
    KontextfreieGrammatik g = new KontextfreieGrammatik();
    g.stringAlsGrammatik(grammatik);
    g.zeigeLinksableitung("aaabbbaabb");
    g.zeigeLinksableitung("abaab");
  }
}
```

Die Ausgabe sieht wie folgt aus. Ist das Wort nicht ableitbar, erfolgt ein Hinweis, gegebenenfalls mit einem ableitbaren Teilwort (hier nicht).

```
S -> TT -> aTbT -> aaTbbT -> aaabbbT -> aaabbbaTb -> aaabbbaabb
abaab ist nicht ableitbar, nur:
```