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

Theoretische Informatik

Sommersemester 2023
4. Aufgabenblatt

Aufgabe 11 (noch eine einfache Turing-Maschine)
Arbeiten Sie weiter mit dem Eclipse-Projekt des vorherigen Aufgabenblatts.
Gegeben Sei das Alphabet={a,b}. Zeigen Sie mit Hilfe einer Turing-Maschine, dass die totale Funktion f: Alphabet* -> Alphabet* mit $f(w) = a^m b^n$ mit anzahl(a,w) =m und anzahl(b,w) = n , also gleiche Anzahl a und b wie im Ausgangswort geordnet, turing-berechenbar ist. Beschreiben Sie Ihren Ansatz mit mindestens 2 Sätzen. Schreiben Sie Ihre Maschine in die Datei beispiele/turingmaschinen/TMABgeordnet.tm. Prüfen Sie Ihr Ergebnis mit test.turingMaschine.ABgeordnetTest.java.

Aufgabe 12 (elementare Nutzung kontextfreier Grammatiken [klausurähnlich], 3+1+2+2 Punkte)
Gegeben sei folgende Grammatik ({Start,A, B},{a, b}, Regeln, Start) mit den Regeln
Start -> AaB A->Aa | a | aa B -> b
a) geben Sie für folgende Worte eine Ableitung an, wenn möglich: aab, b, aaaab
b) Ist die Grammatik mehrdeutig? Begründen Sie ihre Antwort.
c) Welche Sprache wird von der Grammatik erzeugt?
d) Ist die Sprache mehrdeutig? Begründen Sie ihre Antwort.

Aufgabe 13 (kontextfreie Sprache Grammatik erstellen [klausurähnlich], 5+2 Punkte)
Gegeben sei folgende kontextfreie Sprache $L = \{a^n b^m c^{n+m} \mid n,m \geq 0\}$
a) geben Sie eine kontextfreie Grammatik an, die L erzeugt
b) Geben Sie für Ihre Grammatik Ableitungen für folgende Worte an: ε, abcc, aacc, bbcc

Aufgabe 14 (elementare Nutzung kontextfreier Grammatiken [klausurähnlich], 3+1+2+2 Punkte)
Gegeben sei folgende Grammatik ({Start,A, B},{a, b}, Regeln, Start) mit den Regeln
Start -> AB A-> aA | a | B | ε, B -> BB | b
a) geben Sie für folgende Worte eine Ableitung an, wenn möglich: aabb, b, a
b) Ist die Grammatik mehrdeutig? Begründen Sie ihre Antwort.
c) Welche Sprache wird von der Grammatik erzeugt?
d) Ist die Sprache mehrdeutig? Begründen Sie ihre Antwort.

Aufgabe 15 (kontextfreie Sprache Grammatik erstellen [klausurähnlich], 5+2 Punkte)
Gegeben sei folgende kontextfreie Sprache $L = \{a^n b^{n/3} \mid n \geq 0\}$, dabei steht / für ganzzahlige Division
a) geben Sie eine kontextfreie Grammatik an, die L erzeugt
b) Geben Sie für Ihre Grammatik Ableitungen für folgende Worte an: ε, aa, aaab, aaaab

Aufgabe 16 (weiter für kontextfreie Sprache Grammatik erstellen)
Von der Veranstaltungsseite ist ein Eclipse-Projekt erhältlich, dass u. a. die Nutzung und Analyse von kontextfreien Grammatiken erlaubt. Eine kurze Einführung befindet sich am Ende dieses Aufgabenblatts. Lesen Sie diese zuerst.
a) Übertragen Sie Ihre Grammatik aus 13a) in das Format des Tools in die vorhandene Datei beispiele/kontextfreiegrammatik/KFGa_nb_nc_nm.kfg. Prüfen Sie mit den JUnit-Tests aus test.kontextfreieGrammatik. KFGa_nb_nc_nmTest.java ob Ihre Grammatik die enthaltenen Tests besteht. Korrigieren Sie gegebenenfalls Ihre Grammatik.
b) Übertragen Sie Ihre Grammatik aus 15a) in das Format des Tools in die vorhandene Datei beispiele/kontextfreiegrammatik/ KFGa_nb_ndiv3.kfg. Prüfen Sie mit den JUnit-Tests aus test.kontextfreieGrammatik. KFGa_nb_ndiv3Test.java ob Ihre Grammatik die enthaltenen Tests besteht. Korrigieren Sie gegebenenfalls Ihre Grammatik.

c) Erstellen Sie Grammatiken für die folgenden Sprachen in der angegebenen Datei im Unterordner beispiele/kontextfreiegrammatiken und testen Sie mit der angegebenen Testdatei. Als Alphabet soll immer {a, b} genutzt werden.

	Sprache	Datei	Test
I.	{}	KFGleer.kfg	KFGleerTest.java
II.	{ε, aa, bbb}	KFGendlich.kfg	KFGendlichTest.java
III.	$\{a^{2n}b^{3n} \mid n > 0\}$	KFGa_2nb_3n.kfg	KFGa_2nb_3nTest.java
IV.	$\{a^n b^{n\%3} \mid n \geq 0\}$, % für modulo	KFGa_nb_nmod3.kfg	KFGa_nb_nmod3Test.java

Kontextfreie-Grammatiken-Bibliothek

Zum Experimentieren steht Ihnen eine Bibliothek von der Veranstaltungsseite zur Verfügung, mit der kontextfreie Grammatiken ausgeführt und überprüft werden, ob ein Wort abgeleitet werden kann oder nicht. Zur Beschreibung von kontextfreien Grammatiken wird folgendes Format genutzt, hier erklärt an der Grammatik, die $\{a^m b^n \mid m \geq n\}$ erzeugt.

```
% Prozentzeichen am Anfang fuer Kommentar
% Nichtterminalzeichen
N: Start A B
% Terminalzeichen
T: a b
% Startsymbol
S: Start
% Regeln
% mehrere in einer Zeile moeglich, muss aber nicht
Start -> AStartB | /eps
A -> Aa | a
B -> b | /eps
```

Generell müssen Nichtterminale, Terminale, das Start-Symbol und die Regeln angegeben werden. Beginnt eine Zeile mit „%“ handelt es sich um einen Kommentar, diese Zeilen könnten also weggelassen werden. Beginnt eine Zeile mit „N:“ folgen dahinter durch Leerzeichen getrennt die Namen der Nichtterminale, die selbst keine Leerzeichen enthalten dürfen. Beginnt eine Zeile mit „T:“ folgen danach wieder durch Leerzeichen getrennt die Terminale. Eine Randbedingung ist dabei, dass Terminale und Nichtterminale nicht der Präfix eines anderen Elements sein dürfen, so darf es nur eines der Terminal- oder Nichtterminal-Zeichen „O“ und „OK“ geben. Beginnt eine Zeile mit einem „S:“ folgt danach genau der eine Name des Startsymbols, der vorher in einer mit „N:“ markierten Zeile definiert werden musste. Die Regeln werden zeilenweise angegeben. Die generelle Form ist: Nichtterminalzeichen, dann ein Pfeil und dann das abgeleitete Wort aus Nichtterminalen und Terminalen. Mehrere Varianten können in einer Zeile stehen und sind mit einem senkrechten Strich getrennt. Das leere Wort wird mit /eps angegeben.

Ein Turing-Maschine kann entweder aus einer Datei oder als String gelesen werden.

```
KontextfreieGrammatik kfg = new KontextfreieGrammatik();
kfg.dateiEinlesen("kontextfreiegrammatiken\\Spachea_nb_m.kfg");
```

oder

```
String grammatik =
    """
    N: AX BX
    T: a b
    S: AX
    r1:: AX -> BX | a
    r2:: BX -> b
    r3:: BX -> /eps
    """;
KontextfreieGrammatik g = new KontextfreieGrammatik();
g.stringAlsGrammatik(grammatik);
```

Die Grammatik im String zeigt eine weitere optionale Möglichkeit, jeder Zeile mit Regeln einen eindeutigen Namen zuzuordnen. Der Name wird mit einem doppelten Doppelpunkt abgeschlossen.

Die Klasse KontextfreieGrammatik stellt einige meist dokumentierte Methoden zur Verfügung. Die am Anfang wichtigsten sind:

```
boolean erg = g.ableitbar(eingabe); // eingabe ist Wort oder String

Wort erg = g.stringAlsWort(eingabe); // eingabe ist String
```

Soll eine Ableitung schrittweise ausgeführt werden, ist das z. B. wie folgt möglich.

```
public static void main(String[] args) {
  String grammatik =
      """
      N: S B
      T: a b
      S: S
      r1:: S -> aSbaSb | ab
      """;
  KontextfreieGrammatik g = new KontextfreieGrammatik();
  g.stringAlsGrammatik(grammatik);
  g.ausfuehren();
}
```

Im Dialog werden dann alle Regeln angeboten. Falls eine Regel an mehreren Stellen anwendbar ist, kann dies im Folgeschritt geklärt werden. Ein Beispieldialog sieht wie folgt aus. Es ist auch erkennbar, dass die Regeln individuelle Namen erhalten haben. In Klammern steht die aktuelle Länge des abgeleiteten Wortes. Eingaben sind umrandet.

```
Terminale =[a, b]
Nichtterminale = [B, S]
Start = S
 (1) r1_0 :: S -> aSbaSb
 (2) r1_1 :: S -> ab
 (0) abbrechen
aktuelles Wort: S (1)

Welche Regel wollen Sie anwenden: 1
Terminale =[a, b]
Nichtterminale = [B, S]
```

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


```
Start = S
 (1) r1_0 :: S -> aSbaSb
 (2) r1_1 :: S -> ab
 (0) abbrechen
aktuelles Wort: aSbaSb (6)

Welche Regel wollen Sie anwenden: 1
Weches Vorkommen (erstes ist 1): 2
Terminale =[a, b]
Nichtterminale = [B, S]
Start = S
 (1) r1_0 :: S -> aSbaSb
 (2) r1_1 :: S -> ab
 (0) abbrechen
aktuelles Wort: aSbaaSbaSbb (11)

Welche Regel wollen Sie anwenden: 2
Weches Vorkommen (erstes ist 1): 2
Terminale =[a, b]
Nichtterminale = [B, S]
Start = S
 (1) r1_0 :: S -> aSbaSb
 (2) r1_1 :: S -> ab
 (0) abbrechen
aktuelles Wort: aSbaaabbaSbb (12)

Welche Regel wollen Sie anwenden: 2
Weches Vorkommen (erstes ist 1): 1
Terminale =[a, b]
Nichtterminale = [B, S]
Start = S
 (1) r1_0 :: S -> aSbaSb
 (2) r1_1 :: S -> ab
 (0) abbrechen
aktuelles Wort: aabbaaabbaSbb (13)

Welche Regel wollen Sie anwenden: 2
Ergebnis: aabbaaabbaabbb
```

Der spezielle Kommentar %ignorespace führt dazu, dass bei der Angabe der Regeln Leerzeichen zwischen den Zeichen stehen dürfen, was die Lesbarkeit erhöht. Weiterhin werden so Leerzeichen aus Eingabewörtern automatisch entfernt. Sollen trotzdem Leerzeichen genutzt werden, sind das Terminalzeichen /space zu ergänzen und passende Regeln zu definieren. Dies zeigt das folgende Beispiel mit den Nichtterminalzeichen $Optional und $Spaces die dafür sorgen dass beliebig viele bzw. mindestens ein Leerzeichen zu nutzen sind. Das $-Zeichen ist nur ein kleiner Trick auch Großbuchstaben als Terminalzeichen zu ermöglichen, da es so keine Präfix-Problematik gibt. Diese Möglichkeiten werden in der folgenden Grammatik zusammengefasst. Die statisch-polymorphe Methode ableitbar liefert bei Worten automatisch ein Objekt vom Typ Ableitung, das mit isErfolgreich() darauf geprüft werden kann, ob das gesamte Wort abgeleitet werden kann.

```
public static void main(String[] args) {
      String grammatik =
            """
            %ignorespaces
            N: Start $Optional $Spaces
            T: a b /space
            S: Start
            Start -> a Start b | $Spaces
            $Optional -> /eps | $Spaces
            $Spaces -> /space | /space $Spaces
            """;
      KontextfreieGrammatik g = new KontextfreieGrammatik();
      g.stringAlsGrammatik(grammatik);
      Wort w = g.stringAlsWort("aa bb");
      System.out.println("aa bb: " + g.ableitbar(w).isErfolgreich());
      Wort w2 = new Wort("aa bb");
      System.out.println("aa bb: " + g.ableitbar(w2).isErfolgreich());
}
```

Die Ausgabe lautet:

```
aa bb: false
aa bb: true
```

Die erste Ausgabe „false“ beruht darauf, dass bei der Umwandlung des Strings „aa bb“ automatisch die Leerzeichen entfernt wurden, was beim Wort-Konstruktor nicht der Fall ist. Kritisch sollte im Beispiel auffallen, dass das Nichtterminal $Optional in diesem Fall überflüssig ist. Weiterhin wird der Kommentar %ignorespaces statt %ignorespace genutzt, was egal ist, da nur der Anfang übereinstimmen muss.