CYK-Algorithmus

Weiterführende Literatur:

- *Theoretische Informatik – Kontextfreie Sprachen*, Seite 45-75
- Hoffmann, *Theoretische Informatik*, Seite 186-188
- *Wikipedia-Artikel „Cocke-Younger-Kasami-Algorithmus“*

Online-Tools

- `https://www.xarg.org/tools/cyk-algorithm/`

Der Algorithmus ist nach Cocke, Younger, Kasami benannt. Er dient dazu das *Wortproblem* für eine Sprache zu entscheiden, die durch eine CNF-Grammatik (*Chomsky-Normalform*) gegeben ist.

Wortproblem

Chomsky-Normalform

Bilde zu einem Wort w alle Teilwörter der Länge $1, 2, 3, \ldots$ und bestimme die Ableitbarkeit von Variablen. Ist das „Teilwort“ der Länge $|w|$ vom Startsymbol ableitbar, so gehört w zur Sprache.

	a	b	a	b	a
i/j	1	2	3	4	5
1	$t_{1,1}$	$t_{1,2}$	$t_{1,3}$	$t_{1,4}$	$t_{1,5}$
2	$t_{2,1}$	$t_{2,2}$	$t_{2,3}$	$t_{2,4}$	
3	$t_{3,1}$	$t_{3,2}$	$t_{3,3}$		
4	$t_{4,1}$	$t_{4,2}$			
5	$t_{5,1}$				

Funktionsweise des Algorithmus

Wir leiten die erste Tabellenzeile aus den Produktionsregeln ab. Beginnend mit der zweiten Tabellenzeile wählen wir paarweise zwei Tabellenzellen aus und zwar mit folgendem Muster:

Zum Beispiel: $t_{2,1}$

(a) $t_{1,1}$ und $t_{1,2}$

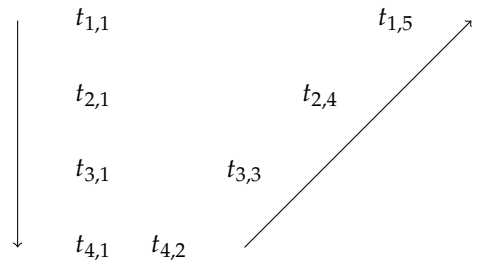
Zum Beispiel: $t_{5,1}$

(a) $t_{1,1}$ und $t_{4,2}$

(b) $t_{2,1}$ und $t_{3,3}$

(c) $t_{3,1}$ und $t_{2,4}$

(d) $t_{4,1}$ und $t_{1,5}$



Sind die richtigen Tabellenzellen ausgewählt, dann kombinieren wir alle Nonterminale von der ersten ausgewählten Zelle mit der zweiten. Dabei muss die Reihenfolge eingehalten werden.

Zum Beispiel: $t_{1,1}$ (A,B) $t_{1,2}$ (C,D): A,C A,D B,C B,D

Steht in der letzten Zelle die Start-Variable, dann ist das Wort ableitbar.

```
// https://github.com/KaustubhD/CYK-implementation-JAVA

/* Example
    S->AB|BC      {"S", "AB", "BC"}
    A->BA|a       {"A", "BA", "a"}
    B->CC|b       {"B", "CC", "b"}
    C->AB|a       {"C", "AB", "a"}

So, the grammer[][] is {
                        {"S", "AB", "BC"},
                        {"A", "BA", "a"},
                        {"B", "CC", "b"},
                        {"C", "AB", "a"}
                       }
*/

import java.util.Scanner;

public class CYKAlgorithmus {
  static int np = 0;
  // Insert grammer here
  static String grammer[][] = { { "S", "AB", "BC" }, { "A", "BA", "a" }, { "B",
   ↪  "CC", "b" }, { "C", "AB", "a" } };

  // Checks if the passed string can be achieved for the grammer
  static String check(String a) {
    String to_ret = "";
    int count = 0;
    for (int i = 0; i < np; i++) {
      for (count = 0; count < grammer[i].length; count++) {
        if (grammer[i][count].equals(a)) {
          to_ret += grammer[i][0];
        }
      }
    }
    return to_ret;
  }

  // Makes all possible combinations out of the two string passed
  static String combinat(String a, String b) {
    String to_ret = "", temp = "";
    for (int i = 0; i < a.length(); i++) {
```

```
      for (int j = 0; j < b.length(); j++) {
        temp = "";
        temp += a.charAt(i) + "" + b.charAt(j);
        to_ret += check(temp);
      }
    }
    return to_ret;
  }

  public static void main(String[] args) {
    String start;
    Scanner in = new Scanner(System.in);
    // Start symbol is generally "S"
    start = "S";
    // np = no of productions
    np = grammer.length;
    System.out.println("Enter the string to be checked >>");
    String str = in.nextLine(), st = "", r = "";
    in.close();
    int count;
    String ans_mat[][] = new String[10][10];

    // Fill the diagnol of the matrix (first iteration of algorithm)
    for (int i = 0; i < str.length(); i++) {
      r = "";
      st = "" + str.charAt(i);
      for (int j = 0; j < np; j++) {
        for (count = 1; count < grammer[j].length; count++) {
          if (grammer[j][count].equals(st)) {
            r += grammer[j][0];
          }
        }
      }
      ans_mat[i][i] = r;
    }

    // Fill the rest of the matrix
    for (int i = 1; i < str.length(); i++) {
      for (int j = i; j < str.length(); j++) {
        r = "";
        for (int k = j - i; k < j; k++) {
          r += combinat(ans_mat[j - i][k], ans_mat[k + 1][j]);
        }
        ans_mat[j - i][j] = r;
      }
    }

    // The last column of first row should have the start symbol
    if (ans_mat[0][str.length() - 1].indexOf(start) >= 0) {
      accept();
    } else {
      reject();
    }

  }

  public static void accept() {
    System.out.println("String is accepted");
    System.exit(0);
  }

  public static void reject() {
```

```
106       System.out.println("String is rejected");
107       System.exit(0);
108     }
109
110   }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/formale_sprachen/CYKAlgorithmus.java`

Literatur

[1] Dirk W. Hoffmann. *Theoretische Informatik*. 2018.

[2] *Theoretische Informatik – Kontextfreie Sprachen*.

[3] *Wikipedia-Artikel „Cocke-Younger-Kasami-Algorithmus“*. `https://de.wikipedia.org/wiki/Cocke-Younger-Kasami-Algorithmus`.