

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, ... 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.

```

3 // https://github.com/KaustubhD/CYK-implementation-JAVA
4
5 /* Example
6   S->AB|BC      {"S", "AB", "BC"}
7   A->BA|a       {"A", "BA", "a"}
8   B->CC|b       {"B", "CC", "b"}
9   C->AB|a       {"C", "AB", "a"}
10
11 So, the grammer[][] is {
12     {"S", "AB", "BC"},
13     {"A", "BA", "a"},
14     {"B", "CC", "b"},
15     {"C", "AB", "a"}
16 }
17 */
18
19 import java.util.Scanner;
20
21 public class CYKAlgorithmus {
22     static int np = 0;
23     // Insert grammer here
24     static String grammer[][] = { { "S", "AB", "BC" }, { "A", "BA", "a" }, { "B",
25     ↪ "CC", "b" }, { "C", "AB", "a" } };
26
27     // Checks if the passed string can be achieved for the grammer
28     static String check(String a) {
29         String to_ret = "";
30         int count = 0;
31         for (int i = 0; i < np; i++) {
32             for (count = 0; count < grammer[i].length; count++) {
33                 if (grammer[i][count].equals(a)) {
34                     to_ret += grammer[i][0];
35                 }
36             }
37         }
38         return to_ret;
39     }
40
41     // Makes all possible combinations out of the two string passed
42     static String combinat(String a, String b) {
43         String to_ret = "", temp = "";
44         for (int i = 0; i < a.length(); i++) {
45             for (int j = 0; j < b.length(); j++) {
46                 temp = "";
47                 temp += a.charAt(i) + "" + b.charAt(j);
48                 to_ret += check(temp);
49             }
50         }
51         return to_ret;
52     }
53
54     public static void main(String[] args) {
55         String start;
56         Scanner in = new Scanner(System.in);

```

```

56 // Start symbol is generally "S"
57 start = "S";
58 // np = no of productions
59 np = grammar.length;
60 System.out.println("Enter the string to be checked >>");
61 String str = in.nextLine(), st = "", r = "";
62 in.close();
63 int count;
64 String ans_mat[][] = new String[10][10];
65
66 // Fill the diagonal of the matrix (first iteration of algorithm)
67 for (int i = 0; i < str.length(); i++) {
68     r = "";
69     st = "" + str.charAt(i);
70     for (int j = 0; j < np; j++) {
71         for (count = 1; count < grammar[j].length; count++) {
72             if (grammar[j][count].equals(st)) {
73                 r += grammar[j][0];
74             }
75         }
76     }
77     ans_mat[i][i] = r;
78 }
79
80 // Fill the rest of the matrix
81 for (int i = 1; i < str.length(); i++) {
82     for (int j = i; j < str.length(); j++) {
83         r = "";
84         for (int k = j - i; k < j; k++) {
85             r += combinat(ans_mat[j - i][k], ans_mat[k + 1][j]);
86         }
87         ans_mat[j - i][j] = r;
88     }
89 }
90
91 // The last column of first row should have the start symbol
92 if (ans_mat[0][str.length() - 1].indexOf(start) >= 0) {
93     accept();
94 } else {
95     reject();
96 }
97
98 }
99
100 public static void accept() {
101     System.out.println("String is accepted");
102     System.exit(0);
103 }
104
105 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](https://github.com/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>.