Praktikum III – CFLs

GTI SoSe 2017 Prof. A. Siebert, T. Franzke

Aufgabe 1.
Sei Λ die Sprache aller Worte über Σ={c, d}, bei denen die Anzahl der c um zwei größer ist als die Anzahl der d, also

$$\Lambda = \{w \mid w \in (c+d)^*,\ |c| = |d| + 2,\ |d| \geq 0\}$$

Zu Λ gehören also z.B. die Worte cc, dccc, cdcc, ccdc, cccd, ddcdcccc.

a. Entwerfen Sie einen NKA, der Λ akzeptiert.

b. Entwerfen Sie eine kontextfreie Grammatik, die Λ generiert.

c. Zeichnen Sie für die Grammatik aus (b.) den Ableitungsbaum von w=cddccc.

d. Wandeln Sie Ihre CFG aus (b.) in CNF um.

e. Verwenden Sie den CYK, um für Ihre CFG aus (d.) zu testen, ob das Wort w=cddccc in Λ ist. Stellen Sie dazu insbesondere die zugehörige V[i,j]-Tabelle auf.

f. Transformieren Sie Ihre CFG aus (b.) mit dem im Skript gegebenen Verfahren in einen NKA.

Aufgabe 2.
Das **Postsche Korrespondenzproblem**, **PCP**, (E. Post, 1897-1954) geht wie folgt:
Gegeben sind n Wortpaare (x_i, y_i) über einem Alphabet Σ.
Gefragt ist, ob es eine Folge $i_1, i_2, ...i_k$ gibt, so dass die Konkatenation der $x_{i_1}\ x_{i_2}\ ...\ x_{i_k}$ gleich der Konkatenation der $y_{i_1}\ y_{i_2}\ ...\ y_{i_k}$ ist.

Bsp.: Sei $\Sigma = \{*, \#, \S\}$ und die Wortpaare gleich $(\#\#,\ \#), (*\#,\ \#*), (\S,\ \#\S)$.
Dann ist die Folge (1, 3) eine Lösung des gegebenen PCP, denn die Zeichenkette ##§ ergibt sich sowohl oben für die x_{i_k} als auch unten für die y_{i_k}:
| §
| #§

Das PCP ist nicht berechenbar! (Versuchen Sie sich intuitiv klar zu machen, warum nicht.) Folgende Ausgänge eines Lösungsversuchs sind also möglich:

- Das PCP hat eine Lösung und diese wird gefunden.
- Das PCP hat keine Lösung und dies wird aufgezeigt.
- Der Lösungsversuch wird nach m Schritten ohne Antwort abgebrochen.

Versuchen Sie, für die folgenden PCPs eine Lösung zu finden oder zu zeigen, dass es keine Lösung geben kann.

a. PCP-a: $(*,\ ***),\ (*\#**,\ *\#),\ (*\#,\ \#)$

b. PCP-b: $(**,\ \#),\ (*\#\#*,\ \#*\#\#),\ (*\#,\ \#*)$

c. PCP-c: $(\#*,\ \#*\#),\ (*\#\#,\ \#\#),\ (\#*\#,\ *\#\#)$

d. PCP-d: $(aba,\ a),\ (bbb,\ aaa),\ (aab,\ abab),\ (bb,\ babba)$

e. PCP-c: $(**\#,\ *),\ (*\#,\ *\#\#),\ (*\#,\ \#*\#),\ (\#*,\ **\#)$

Aufgabe 3.
Implementieren Sie das PCP in Java und lösen Sie (soweit möglich) die obigen PCP-Instanzen (a.)-(e.).

Wenn Ihnen dies als Info ausreicht, dann dürfen Sie sich gerne daran machen, Ihre eigenen Ideen zu implementieren. Ansonsten gebe ich im folgenden ein paar Hinweise, wie ich dieses Problem angegangen bin.

Auf Moodle finden Sie zwei Java-Dateien:

- `BinaryStrings.java` enthält eine rekursive Methode zur Erzeugung aller 2^N N-stelligen 0/1-Zeichenketten.

  Sie verwendet die gleiche Lösungsstrategie wie meine PCP-Implementierung: Ich starte mit einem leeren Lösungsstring und erweitere ihn solange, bis eine Lösung erreicht ist (oder die Methode eine vorgebene Anzahl von Schritten ausgeführt hat).

  Kleine Feinheit: `BinaryStrings.java` ist (doppelt) rekursiv und führt daher eine Tiefensuche (*Depth First Search*) durch. Da das PCP unentscheidbar ist, ist es geschickter, eine Breitensuche durchzuführen. Diese setzt man um, indem man die noch abzuarbeitenden Lösungskandidaten in eine Warteschlange (*queue, First-In-First-Out*) schreibt. Java hat hierzu die Klasse `ArrayDeque` mit den Methoden `add()` bzw. `addLast()` zum Anfügen eines neuen Elementes am Ende der Warteschlange und `remove()` zum Entfernen des ersten Elementes.

- Die Datei `PCP.java` enthält das Gerüst meiner Implementierung. Insbesondere verwende ich eine Hilfsklasse `PostInst`, welche nichts anderes macht als die beiden Teile eines PCP-Wortpaares in einer Datenstruktur zusammen zu fassen. Ich benutze `PostInst` sowohl für die Definition der PCP-Instanz als auch für die Lösung.

Für dieses Praktikum reicht es, die Lösung als Zeichenkette auszugeben, im Beispiel oben also ##§. Schöner wäre eine Ausgabe, welche die Wortpaare sichtbar macht, also
| §
| #§