Quickies

Pumping-Lemma für ~~kontextfreie~~ reguläre Sprachen

Voraussetzung:
Sei L eine ~~kontextfreie~~ reguläre Sprache.

Folgerung:
$\exists$ Pumping-Zahl $p \in \mathbb{N}$ $\quad \forall$ Wort $w \in L$ mit $|w| \geq p$ $\quad \exists$ Aufteilung $w = ~~ab~~cde$:

- Pumping-Garantie 1: $1 \leq |~~b~~d| \leq |~~b~~cd| \leq p$
- Pumping-Garantie 2: $\forall i \in \mathbb{N}_0 : ~~ab~~^i cd^i e \in L$

Aufgabe 1

```
abstract class DEA {
  Knoten[] knoten;
  char[] alphabet;
  abstract function delta(Knoten, char) returns Knoten;
  abstract function changeDelta(Knoten, char, neuerKnoten) returns void;
  Knoten start;
  Knoten[] endzustaende;
}

private function toArray(int x, int y) return int[] {
  if (x > y) return new int[2] {y, x};
  else return new int[2] {x, y};
}

function tablefilling(DEA eingabeDEA) returns DEA {
  int n = eingabeDEA.knoten.length;          // Anzahl der Knoten
  int nz = eingabeDEA.alphabet.length;       // Anzahl Buchstaben in Sigma
  int[][][][] tabelle = new int[n][][][];
```

```
    // tabelle[Zeile][Spalte][Liste][2 für zeile,spalte]
    // wenn angekreuzt, dann [Zeile][Spalte] = NULL

// falls nur quadratische Arrays möglich: tabelle = int[n][n][nz][2];

// Tabelle initialisieren
// und Kreuze setzen bei unterschiedlichen Endzustandsarten.
for (int zeile = 1; zeile < n; zeile++) {   //tabelle[0] hat 0 Einträge.
  tabelle[zeile] = new int[zeile][][];
  for (int spalte = 0; spalte < zeile; spalte++) {
    boolean knoten_i_ist_Endzustand = eingabeDEA.endzustaende.contains(eingabeDEA.knoten[zeile]);
    boolean knoten_j_ist_Endzustand = eingabeDEA.endzustaende.contains(eingabeDEA.knoten[spalte]);
    boolean gleiche_Endzustand = (knoten_i_ist_Endzustand == knoten_j_ist_Endzustand);

    if (gleiche_Endzustand) {
      tabelle[zeile][spalte] = new int[nz][];

      // buchstabe ist der Buchstabenindex in Sigma.
      for (int buchstabe = 0; buchstabe < nz; buchstabe++) {
        int x = eingabeDEA.delta(eingabeDEA.knoten[zeile], eingabeDEA.buchstabe[buchstabe]).index;
        int y = eingabeDEA.delta(eingabeDEA.knoten[spalte], eingabeDEA.buchstabe[buchstabe]).index;
        tabelle[zeile][spalte][buchstabe] = toArray(x, y);
      }
    } else { // !gleiche_Endzustand
      tabelle[zeile][spalte] = null;
    }
  }
}

//Tabelle initialisiert. Jetzt Kreuze setzen.
boolean job = true;                          // job = !fertig.
while(job) {
  job = false;
  for (int zeile = 1; zeile < n; zeile++) {
    for (int spalte = 0; spalte < zeile; spalte++) {
      if (tabelle[zeile][spalte] != null) {
        for (int buchstabe = 0; buchstabe < nz; buchstabe++) {
          int x = tabelle[zeile][spalte][buchstabe][0];
          int y = tabelle[zeile][spalte][buchstabe][1];
```

```
          if (tabelle[x][y] == null) {
             job = true;
             tabelle[zeile][spalte] = null;
             break; //buchstabenarray
          }
        }
      }
    }
  }
}

DEA resultDEA = eingabeDEA;
int delete = 0;                          // Zähler für gelöschte Knoten

for (int zeile = n; zeile < n; zeile++) if (tabelle[zeile] != null) {
  // der Knoten wurde noch nicht gelöscht.

  for (int spalte = 0; spalte < zeile; spalte++) if (tabelle[zeile][spalte] != null) {
  // kein Kreuz

  //biege alle Pfeile, die auf knoten[zeile] zeigen, auf knoten[spalte] um.
      for (int knoten = 0; knoten < n; knoten++) if (resultDEA.knoten[knoten] != null) {
                                          // es reicht, die Knoten zu prüfen, die es noch gibt
        for (int buchstabe = 0; buchstabe < nz; buchstabe++) {
          if (eingabeDEA.delta(eingabeDEA.knoten[knoten], eingabeDEA.buchstabe[buchstabe]) ==
              == eingabeDEA.knoten[zeile]) {
            resultDEA.changeDelta(
             /* from */ eingabeDEA.knoten[knoten], eingabeDEA.buchstabe[buchstabe],
             /* to   */ eingabeDEA.knoten[spalte]);
          }
        }
      }

      //biege ggf. auch den Startpfeil um
      if (resultDEA.start == resultDEA.knoten[zeile]) {
        resultDEA.start = resultDEA.knoten[spalte];
      }

      // lösche Knoten.
```

```
        resultDEA.knoten[zeile] = null;
        delete++;
        // lösche Knotenzeile.
        // Weil wir mit zeilen-for-Schleife von unten begonnen haben,
        //  wird die zugehörige Spalte nicht mehr abgefragt.
        tabelle[zeile] = null;
        break;      // break spalte = continue zeile // aktuelle Zeile wurde gelöscht.

    } // next spalte
  } // next zeile

  if (delete == 0) {
    return eingabeDEA;
  } else {           // null aus knoten-Array entfernen
    Knoten[] knotenmenge = new Knoten[n - delete];

    for (int knoten = 0, knotenNeu = 0; knoten < nz; knoten++) {
      if (resultDEA[knoten] != null) {
        knotenmenge[knotenNeu] = resultDEA[knoten];
        knotenNeu++;
      }
    }
    resultDEA.knoten = knotenmenge;
    return resultDEA;
  }
}
```

Aufgabe 2

gegebene Grammatik:

S ⟶ a A D | A

C ⟶ D | a b d d

B ⟶ b | A | A A

A → a B | b A B | B

D → d | S | C C

Zyklen bestimmen

C → D → S → A ↔ B

Zyklen eliminieren und Umsortieren

C → a A | b A A | b | A A | a A D | d | C C | a b d d

D → a A | b A A | b | A A | a A D | d | C C

S → a A | b A A | b | A A | a A D

A → a A | b A A | b | A A

Chomsky-Normalform-Algorithmus anwenden

C → X A | Y F | b | A A | X G | d | C C | X H

D → X A | Y F | b | A A | X G | d | C C

S → X A | Y F | b | A A | X G

A → X A | Y F | b | A A

X → a

Y → b

Z → d

F → A A

G → A D

H → Y J

J → Z Z

Aufgabe 3

Rekonstruierte Teilgrammatik

S → A B

A → a

B → B R | C B | A B | b

C → c

R → r

CNF / Startsymbol?

Ja, die Grammatik ist in Chomsky-Normalform.
Ja, die Grammatik definiert eindeutig das Startsymbol S.

Ableitungssequenz

S ⇒ AB ⇒ ABR ⇒ ACBR ⇒ ACABR ⇒ ACAABR ⇒ acaabr

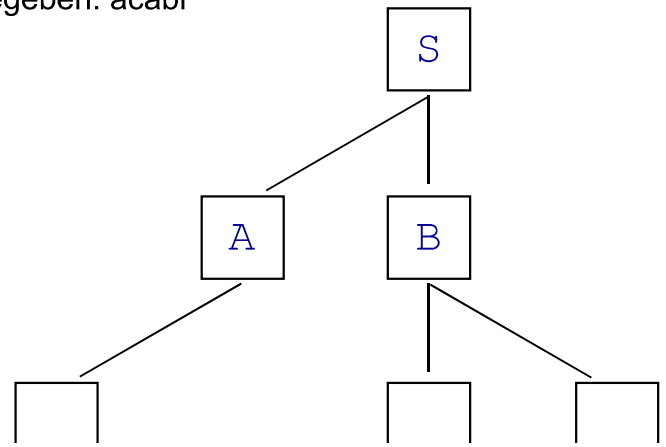
S ⇒ AB ⇒ AAB ⇒ aab

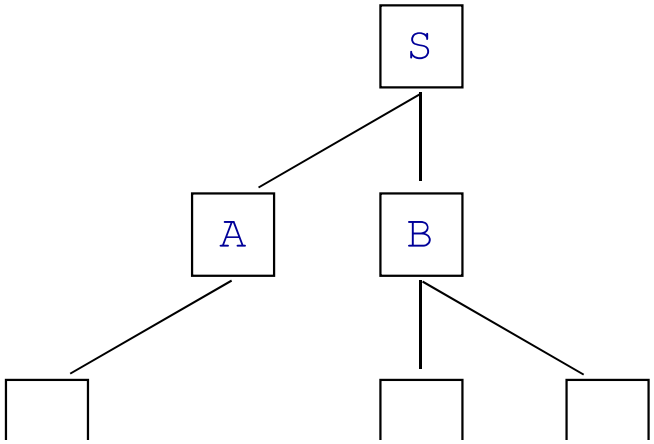
S ⇒ AB ⇒ ABR ⇒ ACBR ⇒ ACABR ⇒ ACABRR ⇒ ACACBRR ⇒ ACACABRR ⇒ acacabrr

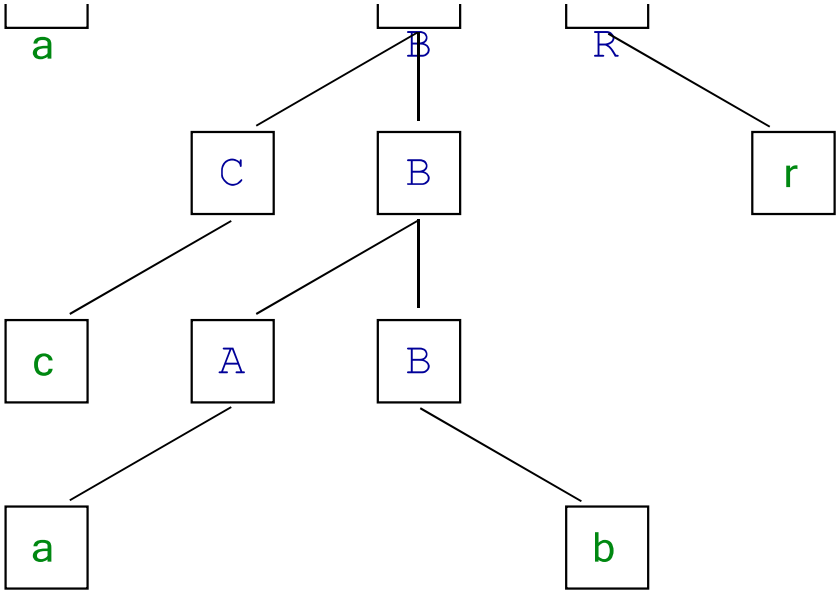
S ⇒ AB ⇒ ABR ⇒ ACBR ⇒ ACABR ⇒ ACACBR ⇒ ACACABR ⇒ acacabr

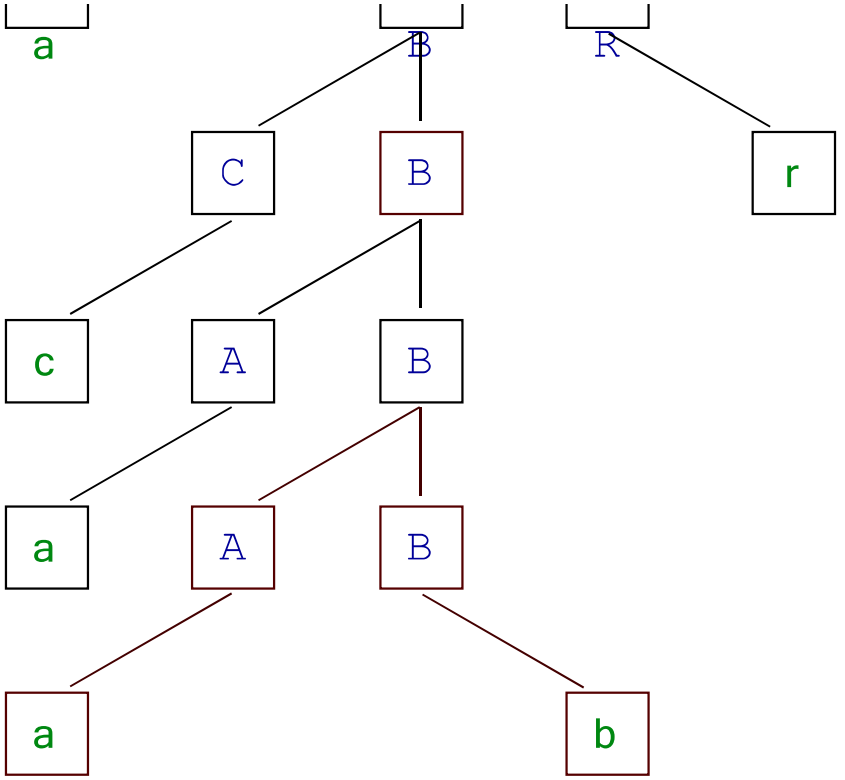
Ableitungsbaum verändern

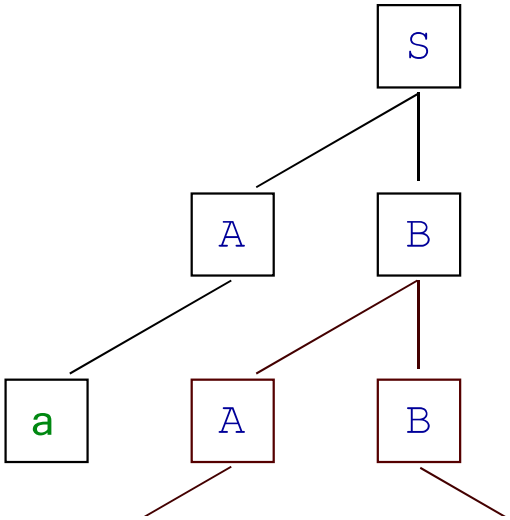
Gegeben: acabr



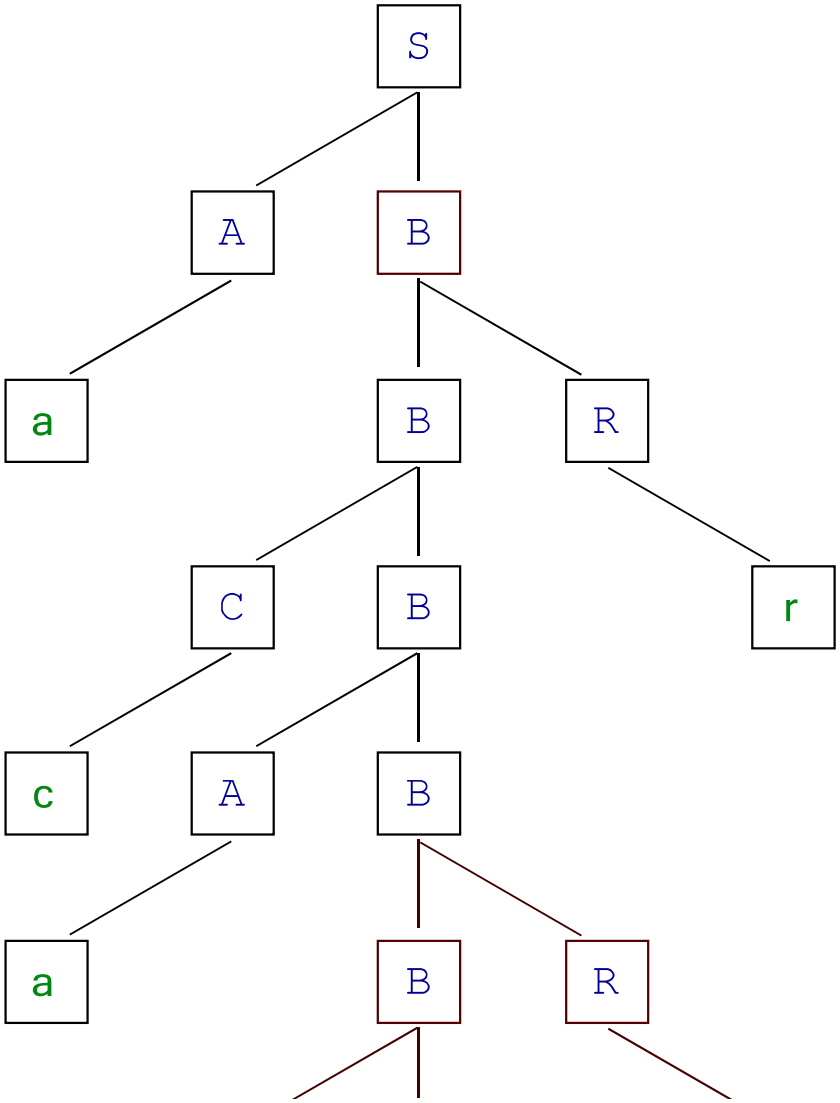


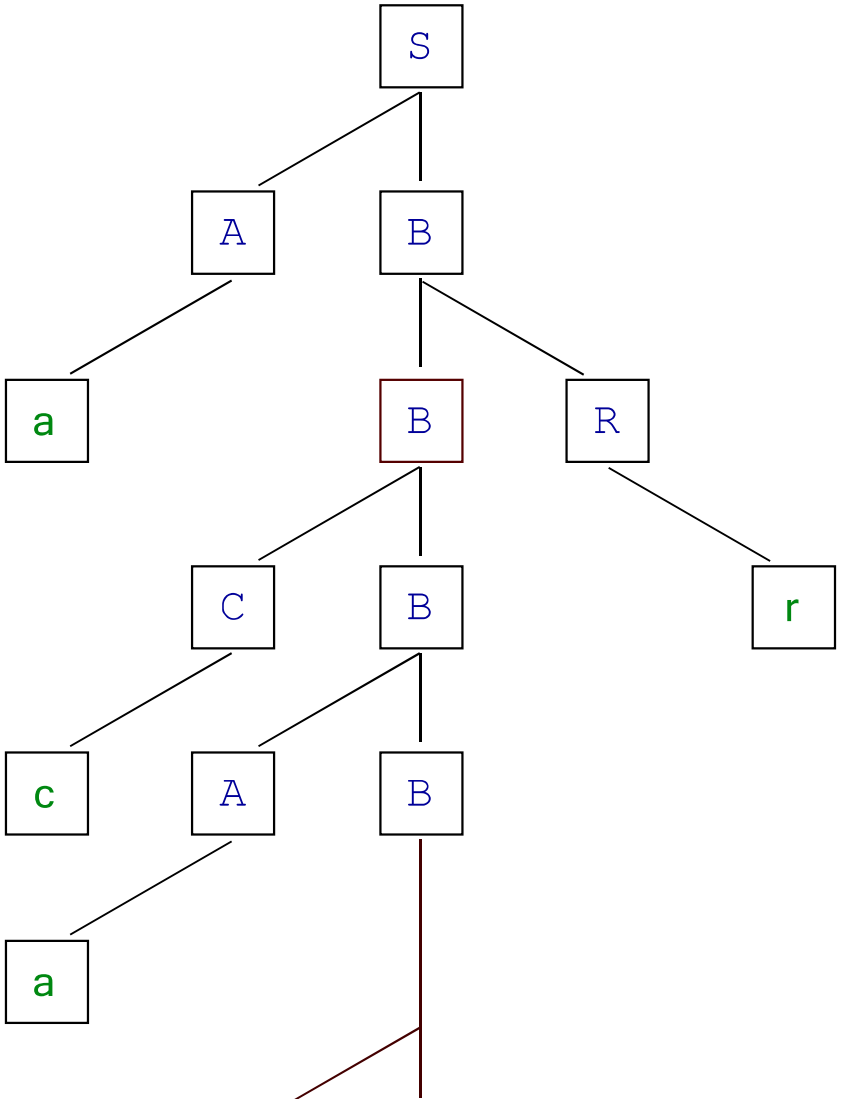

a
B
R
C
B
r
c
A
B
a
b

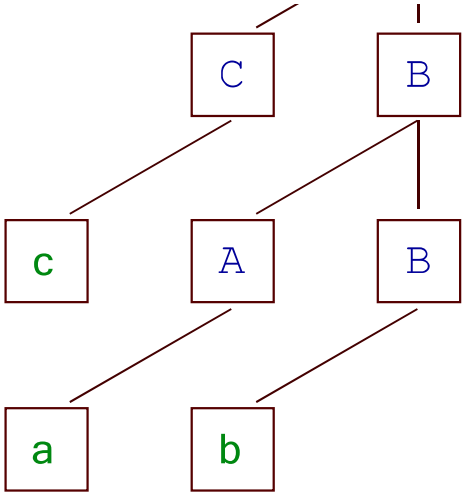

a
B
R
C
B
r
c
A
B
a
A
B
a
b

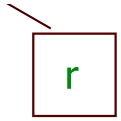

S
A
B
a
A
B

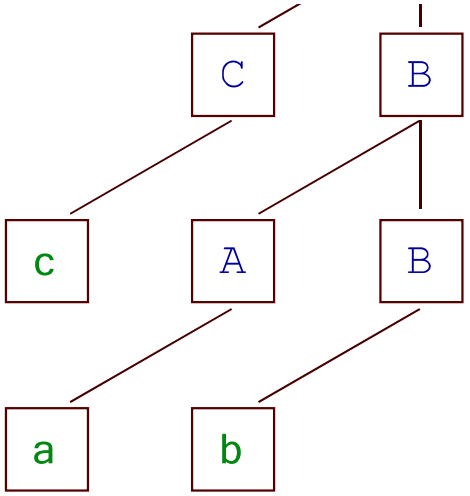

a
b


S
A
B
a
B
R
C
B
r
c
A
B
a
B
R


S
A
B
a
B
R
C
B
r
c
A
B
a


C
B
c
A
B
a
b


r


C
B
c
A
B
a
b