RWTH Aachen **SS 2017**
Lehrgebiet Theoretische Informatik Blatt 6
Rossmanith–Dreier–Hark–Kuinke 12.6.2017

Lösungsvorschlag zur Vorlesung Formale Sprachen, Automaten und Prozesse

Aufgabe T18

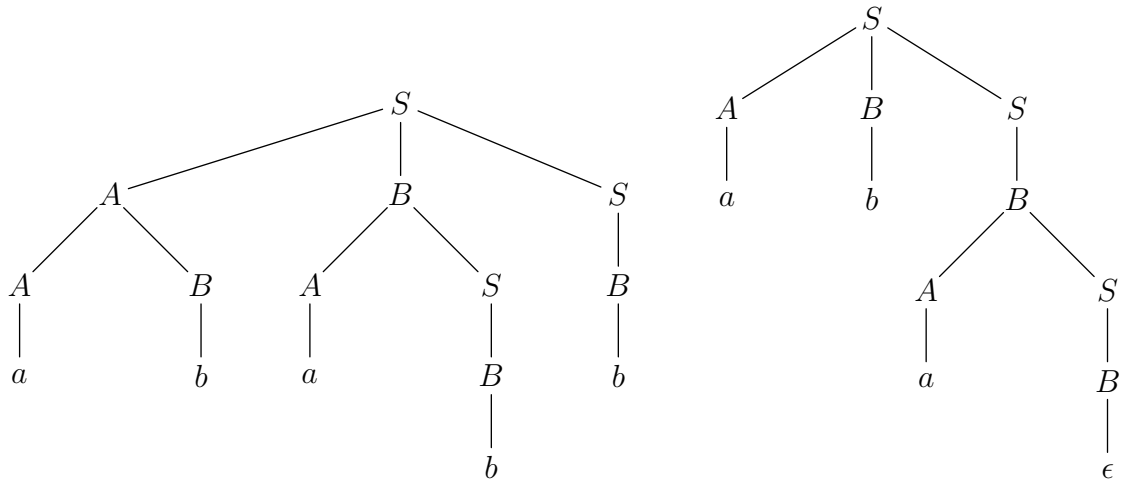
a) G_1: $S \to aSb \mid \epsilon$

b) G_2: $S \to aSa \mid bSb \mid a \mid b \mid \epsilon$

c) G_3: $S \to aSa \mid bSb \mid aAb \mid bAa,\ A \to aA \mid bA \mid \epsilon$

d) G_4: $S \to SS \mid$ `<b>`S`</b>` $\mid$ `<i>`S`</i>` $\mid aS \mid bS \mid \ldots \mid zS \mid 0S \mid \ldots \mid 9S \mid \epsilon$

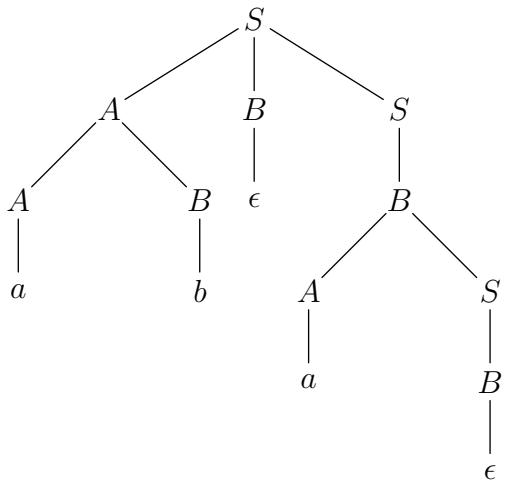
Aufgabe T19



$S \Rightarrow ABS \Rightarrow aBS \Rightarrow abS \Rightarrow abB \Rightarrow abAS \Rightarrow abaS \Rightarrow abaB \Rightarrow aba$

$S \Rightarrow ABS \Rightarrow ABBS \Rightarrow aBBS \Rightarrow abBS \Rightarrow abASS \Rightarrow abaSS \Rightarrow abaBS \Rightarrow\Rightarrow ababS \Rightarrow ababB \Rightarrow ababb$.

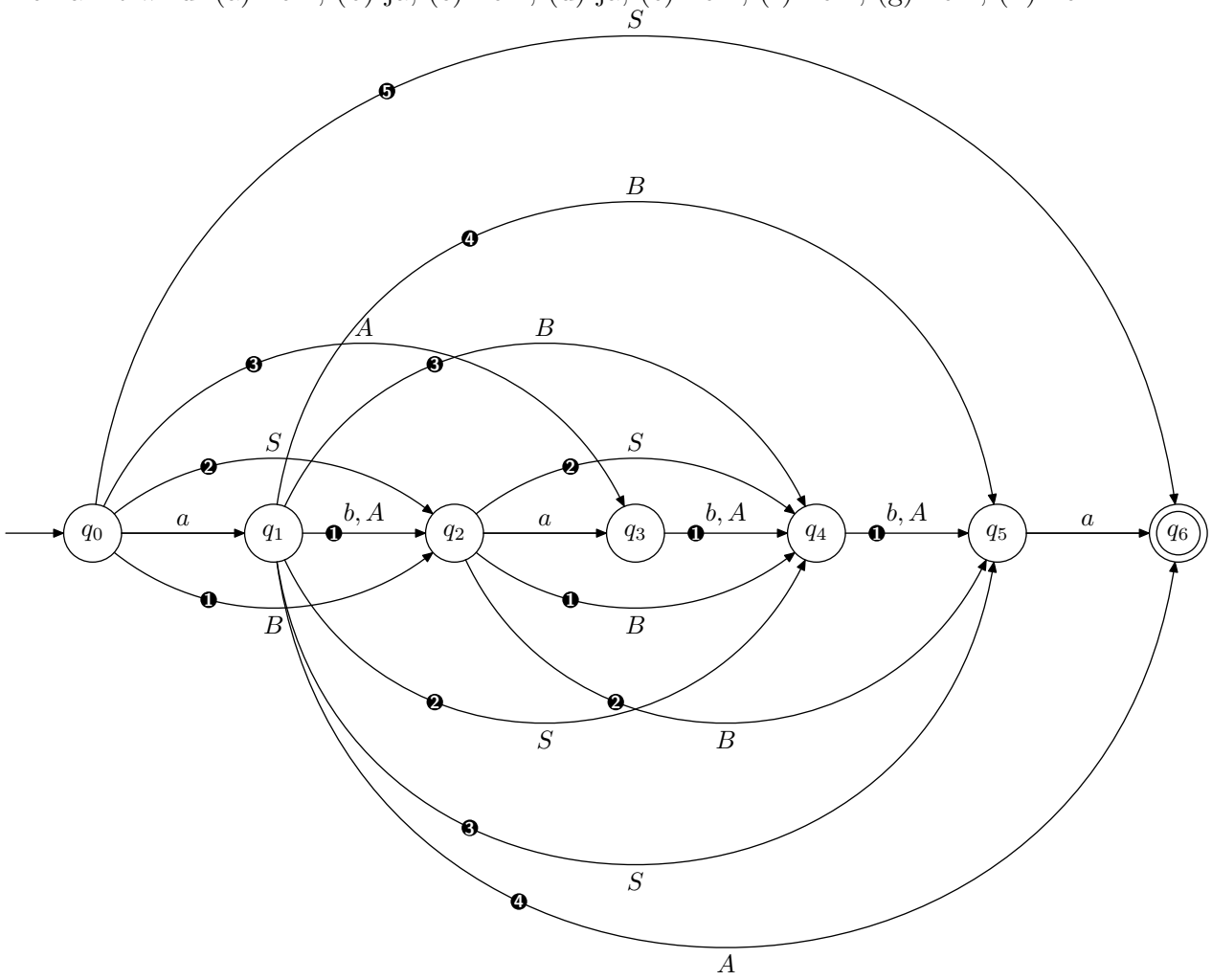
Die Grammatik ist nicht eindeutig, da man aba auch so ableiten könnte:



Aufgabe T20

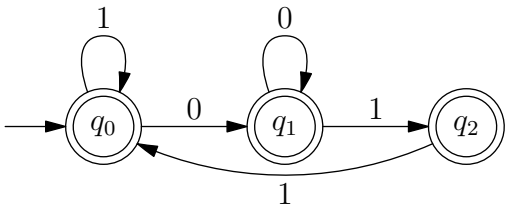
Es gilt G kann *ababba* erzeugen genau dann, wenn $S \in pre^*_G(\{ababba\})$. Wir nehmen $L = \{ababba\}$ und den sich einfach ergebenden NFA mit sieben Zuständen, der L erkennt. Dann konstruieren wir den Automaten für $pre^*_G(L)$ mit Hilfe des Algorithmus aus der Vorlesung. Die Sättigung tritt nach fünf Durchgängen ein.

In der letzten Sättigungsrunde ergibt sich ein mit S beschriebener Übergang vom Start- in den Endzustand. Das bedeutet, daß das Startsymbol S in $pre^*_G(\{ababba\})$ enthalten ist. Folglich kann G auch das Wort *ababba* erzeugen. Die acht Nebenaufgaben sind dann schnell erledigt. α ist auf *ababba* ableitbar genau dann wenn es von dem Automaten erkannt wird. (a) nein, (b) ja, (c) nein, (d) ja, (e) nein, (f) nein, (g) nein, (h) nein.

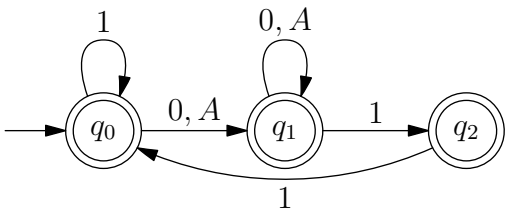


Aufgabe H12 (10 Punkte)

Wir konstruieren zunächst einen NFA für L:



Als nächstes führen wir alle möglichen Sättigungsschritte durch:



Nach der Sättigung des NFA zeigt sich, daß $pre^*_G(L)$ die Sprache aller Wörter über $\{0, 1, A\}^*$ ist, die weder 010 noch 01A noch $A10$ noch $A1A$ enthalten. Weil $A01AA1100$ das Teilwort 01A enthält, kann es folglich gar kein α geben, das von $A01AA1100$ her abgeleitet werden kann.
Weiterhin ist $L(G) \cap L$ leer, weil S auf keiner Transition des Automaten vorkommt und damit auch insbesondere nicht vom Start- zu einem Endzustand führt, was bedeutet, daß man von der Grammatik G zu keinem Wort aus L ableiten kann.

Aufgabe H13 (15 Punkte)
Die leichtere Aufgabe läßt sich beispielsweise mit dem Java-Programm in Abbildung 1 lösen.
Als Beispiel liest es das Wort `while(bla) bla-zock*25-23;` und erzeugt daraus folgende Liste von *Token*:

```
<while>
<(>
<ID:bla>
<)>
< >
<ID:bla>
<->
<ID:zock>
<*>
<NUM:25>
<->
<NUM:23>
<;>
```

Das Programm geht folgendermaßen vor: Seien $w_1, \ldots, w_n$ die Wörter, welche die einfachen regulären Ausdrücke darstellen. Wir konstruieren einen DFA, dessen Zustände alle Präfixe aller w_i sind. Sind u und ua zwei solche Präfixe, deren Länge sich genau um eins unterscheiden, dann gibt es die Transition $q_u a\, q_{ua}$. Fehlende ausgehende Transitionen von q_u werden so ergänzt: Falls u alphanumerisch ist, dann führen alphanumerische Transitionen zum Zustand q_{id}. Falls u numerisch ist, dann führen numerische Transitionen zu q_{num}. Andere Transitionen führen zu einem Fangzustand. Die Transitionen an q_{id} und q_{num} werden wie erwartet definiert.

Aufgabe H14 (7 Punkte)
Wir können den pre^*-Automaten aus Aufgabe T20 verwenden um die Lösung zu finden, da die beiden Wörter Unterwörter von *ababba* sind. Wir müssen nur entsprechende Zustände entfernen und Start und Endzustand neu setzen. Für a) z.B. q_0 und q_6 löschen, und q_1 wird Start- und q_5 wird Endzustand.

```
import java.util.*;

class Token {
  String text, name;
  public int hashCode() { return name.hashCode(); }
  public boolean equals(Object obj) {
    if(!(obj instanceof Token)) { return false; }
    Token other = (Token) obj;
    return name.equals(other.name);
  }
  Token(String n, String w) { text = w; name = n; }
  Token(String n) { this(n, n); }
  public String toString() {
    if(name.equals(text)) { return name; }
    else { return"[" + name + ":" + text + "]"; }
  }
}
public class Scanner {
  Set<String> dictionary = new HashSet<String>();
  NFA<String, Character> M = null;
  Set<Character> alpha = alphacharacters();
  Set<Character> alphacharacters() {
    Set<Character> alpha = new HashSet<Character>();
    for(char a = 'a'; a ≤ 'z'; a++) { alpha.add(a); }
    for(char a = 'A'; a ≤ 'Z'; a++) { alpha.add(a); }
    return alpha;
  }
  private boolean alphabetic(String w) {
    for(int i = 0; i < w.length(); i++) {
      if(!alpha.contains(w.charAt(i))) { return false; }
    }
    return true;
  }
  public void addWord(String word) {
    dictionary.add(word);
  }
  private void constructNFA() {
    Set<String> states = new HashSet<String>();
    // states are all prefixes of all words in dictionary
    for(String word : dictionary) {
      for(int i = 0; i ≤ word.length(); i++) {
        states.add(word.substring(0, i));
      }
    }
    states.add("ID"); states.add("NUM");
    M = new NFA<String, Character>(states);
    for(String q : states) {
      if(q.length() > 0 &&
          !q.equals("ID") && !q.equals("NUM")) {
        String p = q.substring(0, q.length() − 1);
        char a = q.charAt(q.length() − 1);
        M.addTransition(p, a, q);
      }
    }
    for(String q : states) {
      if(alphabetic(q)) {
        Set<Character> inDelta = M.delta.get(q).keySet();
        for(char a : alpha) {
          if(!inDelta.contains(a)) {
            M.addTransition(q, a, "ID");
          }
        }
      }
    }
    for(char c = '0'; c ≤ '9'; c++) {
      M.addTransition("", c, "NUM");
      M.addTransition("NUM", c, "NUM");
    }
  }
  private String element(Set<String> qset) {
    for(String q : qset) { return q; }
    return null;
  }
  public Token firstLongestMatch(String w) {
    if(M == null) { constructNFA(); }
    String u = w + "$";
    Set<String> qset = new HashSet<String>();
    String latest = null;
    qset.add("");
    int i, latestindex = 0;
    for(i = 0; !qset.isEmpty(); i++) {
      if(dictionary.contains(element(qset))
        || element(qset).equals("ID")
        || element(qset).equals("NUM")) {
        latest = element(qset);
        latestindex = i;
      }
      qset = M.simulateOneStep(qset, u.charAt(i));
    }
    if(latest ≠ null) {
      return new Token(latest, w.substring(0, latestindex));
    }
    return new Token("error", w);
  }

  public List<Token> tokenize(String w) {
    List<Token> result = new LinkedList<Token>();
    String u = w;
    while(u.length() > 0) {
      Token tok = firstLongestMatch(u);
      u = u.substring(tok.text.length());
      result.add(tok);
    }
    return result;
  }
  static public void main(String args[]) {
    Scanner scanner = new Scanner();
    scanner.addWord(" "); scanner.addWord("while");
    scanner.addWord("if"); scanner.addWord("(");
    scanner.addWord(")"); scanner.addWord(";");
    scanner.addWord("+"); scanner.addWord("-");
    scanner.addWord("*");
    String source = "while(bla) bla-zock*25-23;";
    List<Token> toks = scanner.tokenize(source);
    System.out.println(toks);
  }
}
```

Abbildung 1: Programm zu Aufgabe H13

a) Es gilt $babb \in L(G)$, da man mit einem S von q_1 nach q_5 kommen kann.

b) Es gilt $babba \notin L(G)$, da man mit einem S *nicht* von q_1 nach q_6 kommen kann.

Die Richtigkeit der Aussagen folgt aus der Richtigkeit des pre^*-Automaten aus Aufgabe T20.

Aufgabe H15 (viele Bonuspunkte)

Die Grammatik ist nicht eindeutig, da es zwei Linksableitungen für das Wort $abaabab$ $aabaababaababbababbbabaabababaabbababaaabbabababababbabbbabccaccbcbcabbbcbaabbcbab$ $bcaabccbbcbbabbababbab$ gibt:

$$\begin{aligned}
S &\Rightarrow A\\
&\Rightarrow abAb\\
&\Rightarrow abaAab\\
&\Rightarrow abaabAbab\\
&\Rightarrow abaababAbbab\\
&\Rightarrow abaababaAabbab\\
&\Rightarrow abaababaabAbabbab\\
&\Rightarrow abaababaabaAababbab\\
&\Rightarrow abaababaabaabAbababbab\\
&\Rightarrow abaababaabaababAbbababbab\\
&\Rightarrow abaababaabaababaAabbababbab\\
&\Rightarrow abaababaabaababaabAbabbababbab\\
&\Rightarrow abaababaabaababaababAbbabbababbab\\
&\Rightarrow abaababaabaababaababbAcbbabbababbab\\
&\Rightarrow abaababaabaababaababbabAbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababAbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbAcbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbAccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabAbccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaAabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaaAaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabAcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaababAbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababAbbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaAabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabAbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbAcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbabAbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababAbbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaAabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaAaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabAbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbAcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabAbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbababAbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababAbbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababaAabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbababababAcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababAbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbAcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabAbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabbAcbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabbbAccbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabbbaAaccbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabbbabccaccbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab
\end{aligned}$$

$$\begin{aligned}
S &\Rightarrow B\\
&\Rightarrow aBb\\
&\Rightarrow abBab\\
&\Rightarrow abaBbab\\
&\Rightarrow abaaBbbab\\
&\Rightarrow abaabBabbab\\
&\Rightarrow abaabaBbabbab\\
&\Rightarrow abaababBababbab\\
&\Rightarrow abaababaBbababbab\\
&\Rightarrow abaababaaBbbababbab\\
&\Rightarrow abaababaabBabbababbab\\
&\Rightarrow abaababaabaBbabbababbab\\
&\Rightarrow abaababaabaaBbbabbababbab\\
&\Rightarrow abaababaabaababBcbbabbababbab\\
&\Rightarrow abaababaabaababaBbcbbabbababbab\\
&\Rightarrow abaababaabaababaaBbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababBcbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbabBccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbabaBbccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababBabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbBaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabBcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaBbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaaBbbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabBabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabaBbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababBcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaBbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaaBbbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabBabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbBaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbaBbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababBcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaBbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaBbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaaBbbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabBabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabBcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabaBbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababBcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababaBbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbababababababBcbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabBccbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabbBaccbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab\\
&\Rightarrow abaababaabaababaababbababbbabaabababaabbababaaabbabababababbabbbabccaccbcbcabbbcbaabbcbabbcaabccbbcbbabbababbab
\end{aligned}$$