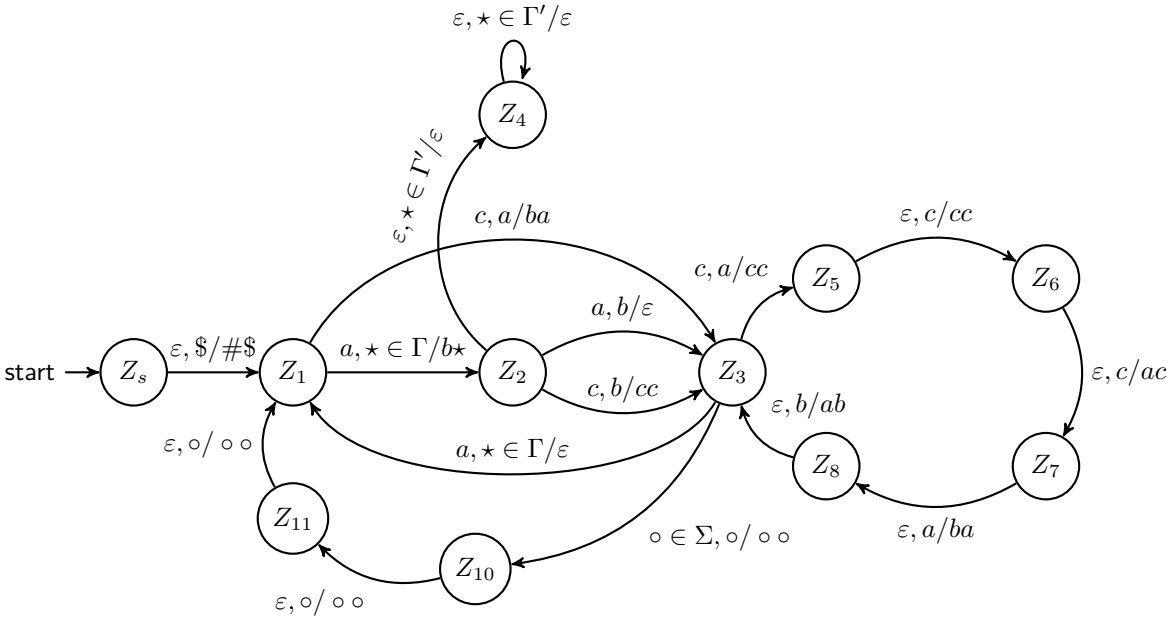
Informatik Ⅽ: – Blatt 6

Rasmus Diederichsen

25. Juli 2014

Aufgabe 6.1

Es sei $\Gamma' = \Gamma \cup \{\$\}$.



Aufgabe 6.2

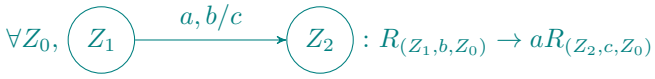
Schritt 1

$$\forall Z \in \mathcal{Z} : S \to R_{(Z_{start}, \#, Z)}$$

Es resultieren die Regeln

$$\boxed{S \to R_{(Z_1, \#, Z_1)} \quad S \to R_{(Z_1, \#, Z_2)}}$$

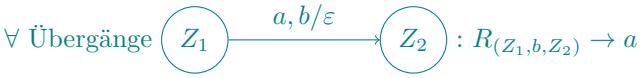
Schritt 2



Es resultieren die Regeln

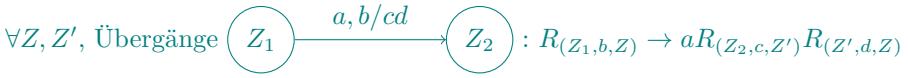
$$\begin{array}{ll} R_{(Z_1,b,Z_1)} \to bR_{(Z_1,b,Z_1)} & R_{(Z_1,b,Z_2)} \to bR_{(Z_1,b,Z_2)} \\ R_{(Z_2,b,Z_1)} \to cR_{(Z_2,b,Z_1)} & R_{(Z_2,b,Z_2)} \to cR_{(Z_2,b,Z_2)} \\ R_{(Z_2,b,Z_1)} \to bR_{(Z_1,c,Z_1)} & R_{(Z_2,b,Z_2)} \to bR_{(Z_1,c,Z_2)} \end{array}$$

Schritt 3



Es resultieren die Regeln

$$\begin{array}{ll} R_{(Z_1,b,Z_1)} \to b & R_{(Z_1,\#,Z_1)} \to b \\ R_{(Z_1,c,Z_1)} \to c & \end{array}$$



Gemäß $\forall X, Y : R_{(Z_1,\#,X)} \to aR_{(Z_2,b,Y)}R_{(Y,\#,X)}$ (es gibt im Graphen nur einen solchen Übergang) resultieren die Regeln

$$\begin{array}{l} R_{(Z_1,\#,Z_1)} \to aR_{(Z_2,b,Z_1)}R_{(Z_1,\#,Z_1)} \\ R_{(Z_1,\#,Z_1)} \to aR_{(Z_2,b,Z_2)}R_{(Z_2,\#,Z_1)} \\ R_{(Z_1,\#,Z_2)} \to aR_{(Z_2,b,Z_1)}R_{(Z_1,\#,Z_2)} \\ R_{(Z_1,\#,Z_2)} \to aR_{(Z_2,b,Z_2)}R_{(Z_2,\#,Z_2)} \end{array}$$

Der gesamte Regelsatz ist also

$S \to R_{(Z_1,\#,Z_1)}$	$S \to R_{(Z_1,\#,Z_2)}$
$R_{(Z_1,b,Z_1)} \to bR_{(Z_1,b,Z_1)}$	$R_{(Z_1,b,Z_2)} \to bR_{(Z_1,b,Z_2)}$
$R_{(Z_2,b,Z_1)} \to cR_{(Z_2,b,Z_1)}$	$R_{(Z_2,b,Z_2)} \to cR_{(Z_2,b,Z_2)}$
$R_{(Z_2,b,Z_1)} \to bR_{(Z_1,c,Z_1)}$	$R_{(Z_2,b,Z_2)} \to bR_{(Z_1,c,Z_2)}$
$R_{(Z_1,b,Z_1)} \to b$	$R_{(Z_1,\#,Z_1)} \to b$
$R_{(Z_1,c,Z_1)} \to c$	
$R_{(Z_1,\#,Z_1)} \to aR_{(Z_2,b,Z_1)}R_{(Z_1,\#,Z_1)}$	
$R_{(Z_1,\#,Z_1)} \to aR_{(Z_2,b,Z_2)}R_{(Z_2,\#,Z_1)}$	
$R_{(Z_1,\#,Z_2)} \to aR_{(Z_2,b,Z_1)}R_{(Z_1,\#,Z_2)}$	
$R_{(Z_1,\#,Z_2)} \to aR_{(Z_2,b,Z_2)}R_{(Z_2,\#,Z_2)}$	

Regeln, die tatsächlich etwas produzieren, sind aquamarin markiert. Regeln, die etwas produzieren könnten, von der Startvariable aus aber nicht erreichbar sind, sind grau hinterlegt. Wir kürzen ab

$$
\begin{aligned}
R_1 &:= R_{(Z_1,\#,Z_1)} \\
R_2 &:= R_{(Z_1,c,Z_1)} \\
R_3 &:= R_{(Z_1,b,Z_1)} \\
R_4 &:= R_{(Z_2,b,Z_1)}
\end{aligned}
$$

und destillieren hieraus

$$
\begin{aligned}
S &\to R_1 \\
R_1 &\to b \mid aR_4R_1 \\
R_2 &\to c \\
R_4 &\to cR_4 \mid bR_2
\end{aligned}
$$

und notieren, dass R_3 nicht erreichbar ist.

Aufgabe 6.3

Der (vermutlich falsche) Algorithmus läuft in $\mathcal{O}\left(|V|^2\right)$ da für jeden Knoten die Suche gestartet wird und in deren Verlauf alle anderen Knoten maximal zweimal besucht werden.

```
1 class DeCycle {
2    static void deCycle(Graph g)
3    {
```

```
4       for(Vertex v : g.getVertices()) v.visited = false;
5       for(Vertex v : g.getVertices()) traverse(v, new ArrayList<
            Vertex>());
6    }
7    static List<Vertex> getTargets(List<Arc>)
8    {
9       /* ... map Arcs to their targets */
10   }
11   static void print(Graph g)
12   {
13      /* output graph */
14   }
15   static void traverse(Graph g, Vertex v, List<Vertex> path)
16   {
17      path.add(v);
18      if(v.visited)
19      contractCycle(path.sublist(path.indexOf(v), path.length()));
20      else
21      {
22         v.visited = true;
23         for (Vertex neighbour : getTargets(g.getOutgoingArcs(v)))
24         traverse(g, neighbour, new ArrayList<Vertex>(path));
25      }
26      v.visited = false; // reset for next search
27   }
28   static void main(String[] argv)
29   {
30      Graph g = new RandomGraph(Integer.parseInt(argv[0]));
31      DeCycle.deCycle(g);
32      DeCycle.print(g);
33   }
34}
```

Aufgabe 6.4

Die Ausgangsgrammatik ist

G

$$
\begin{array}{lll}
S \to D & C \to eEd & E \to C \\
S \to dA & C \to B & \\
A \to aS & D \to CdDeB & \\
A \to S & D \to A & \\
A \to E & D \to d & \\
B \to db & E \to B & \\
B \to C & E \to BB &
\end{array}
$$

Schritt 1: Erzeuge Regeln $A_x \to x$, ersetze $a, \ldots, e$ in G durch $A_a, \ldots, A_e$.

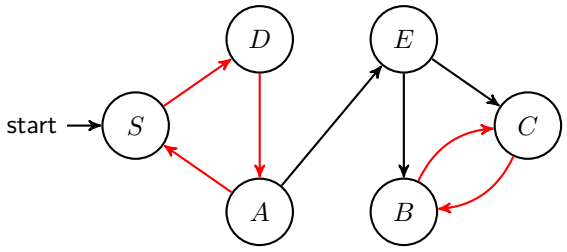
G'

$S \to D$	$C \to A_e E A_d$	$E \to C$
$S \to A_d A$	$C \to B$	$A_a \to a$
$A \to A_a S$	$D \to C A_d D A_e B$	$A_b \to b$
$A \to S$	$D \to A$	$A_d \to d$
$A \to E$	$D \to d$	$A_e \to e$
$B \to A_d A_b$	$E \to B$	
$B \to C$	$E \to BB$	

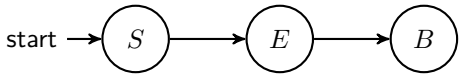
Schritt 2: Entferne Regeln der Form $V \to V$. Folgende Regeln sind zu eliminieren:

$S \to D$	$C \to A_e E A_d$	$E \to C$
$S \to A_d A$	$C \to B$	$A_a \to a$
$A \to A_a S$	$D \to C A_d D A_e B$	$A_b \to b$
$A \to S$	$D \to A$	$A_d \to d$
$A \to E$	$D \to d$	$A_e \to e$
$B \to A_d A_b$	$E \to B$	
$B \to C$	$E \to BB$	

Wir erstellen den zugehörigen Graphen.



Es existieren Zyklen $S \to D \to A \to S$ und $B \to C \to B$ bzw. $C \to B \to C$. Der kontrahierte Graph ist



Wir eliminieren die Senken beginnend mit B und transformieren so die Regelmenge $\{S \to E, E \to B\}$.

$E \to B$ wird zu

$$E \to A_d A_b$$
$$E \to A_e E A_d$$

$S \to E$ wird zu

$$\left.\begin{array}{l} S \to BB \\ S \to A_d A_b \\ S \to A_e E A_d \end{array}\right\} \text{vorherige Elimination}$$

Die neue Grammatik ist

G''

$$S \to A_d S \mid A_a S \mid BB \mid A_d A_b \mid A_e E A_d \mid B A_d S A_e B \mid d$$
$$B \to A_d A_b \mid A_e E A_d$$
$$E \to A_d A_b \mid A_e E A_d \mid BB$$
$$A_a \to a$$
$$A_b \to b$$
$$A_d \to d$$
$$A_e \to e$$

Schritt 3: Kürzung zu langer Regeln. Wir führen für Regeln $V \to V_1, \ldots, V_k$ mit $k > 2$ Regeln $V \to V_1 U_1, U_1 \to V_2 U_2, \ldots, U_k \to V_{k-1} V_k$ ein und erhalten so

G'''

$$S \to A_dS \mid A_aS \mid BB \mid A_dA_b \mid A_eU_1 \mid BU_2$$
$$U_1 \to EA_d$$
$$U_2 \to A_dU_3$$
$$U_3 \to SU_4$$
$$U_4 \to A_eB$$
$$B \to A_dA_b \mid A_eU_1$$
$$E \to A_dA_b \mid A_eU_1 \mid BB$$
$$A_a \to a$$
$$A_b \to b$$
$$A_d \to d$$
$$A_e \to e$$

Diese Grammatik ist nun in Chomsky Normal Form.

Aufgabe 6.5

Man nehme an, das Pumping Lemma gilt, n bezeichne die zugehörige Wortmindestlänge. Sei $z = 1^n21^n21^n$. Es existiert nach Vorraussetzung eine Zerlegung $uvwxy$ von z, sodass $|vx| \geq 1, |vwx| \leq n$ und $uv^iwx^iy \in L \forall i$

Die Fälle, in denen v, x verschiedene Symbole beinhalten (also sowohl 1en als auch 2en), braucht man gar nicht zu betrachten, da auf diese Weise Teilwörter der Form $(1^k2)^i$ o.Ä. entstehen können, was natürlich nicht für alle i in L enthalten ist.

Es gibt vier Möglichkeiten, aus welchen Zeichen vwx bestehen kann.

1. $vwx = 1^k, k \leq n$. In diesem Fall kann durch Pumpen eine beliebige Anzahl 1en erzeugt werden, $uv^iwx^iy = 1^l21^n21^n$ für $l \neq n \notin L$.

2. $vwx = 1^k2, k \leq n-1$. In diesem Fall können durch Pumpen beliebig viele 1en erzeugt werden falls $|v| > 0$ und eventuell 2en, falls $|x| > 0$. Daher $uv^iwx^iy \notin L$.

3. $vwx = 1^k21^l, 1 \leq k + l \leq n-1$. In diesem Fall können durch Pumpen beliebig viele 1en erzeugt werden, eventuell auch beliebig viele 2en, falls $|v| > 1$ oder $|x| > 1$. Daher $uv^iwx^iy \notin L$, da der letzte Teil 1^n nicht mitgepumpt wird.

4. $vwx = 21^k, k \leq n-1$. In diesem Fall können durch Pumpen beliebig viele 1en erzeugt werden, eventuell 2en, falls $|v| \geq 1$. Mithin $uv^iwx^iy \notin L$.

In keinem Fall ist $uv^iwx^iy \in L$, was einen Widerspruch zur Annahme darstellt.

Aufgabe 6.6

Bei der Umwandlung eines NDKA-AdLK in eine kontextfreie Grammatik können Regeln der folgenden Arten generiert werden

1. $V \to \Sigma$
2. $V \to V$
3. $V \to \Sigma \times V$
4. $V \to \Sigma \times V \times V$
5. $(V \to \varepsilon)$

Dieses Problem ist nur dann lösbar, wenn die Grammatik nicht linksrekursiv ist (also kein Regeln der Form $A \to AB$ enthält).

Um die Grammatik in GNF zu überführen, geht man wie folgt vor:

```
1  entferne Kreise in Regeln V_i -> V_j
2  while exists Regel A -> V_i
3     forall Regeln V_i -> *
4        kreiere Regel A -> rhs(V_i)
5     end
6     entferne A -> V_i
7     entferne Regel V_i -> * falls !exists Regel R := V -> V* mit V_i in
           rhs(R)
8  end
9  while exist Regel A -> V_1V_2
10    forall Regeln V_1 -> * kreiere Regel A -> rhs(V_1)V_2
11    entferne Regel A-> V_1V_2
12 end
```