Zusammenfassung zur Vorlesung Basismodul Computerlinguistik

Tarjan-Tabelle &
Einfügen und Löschen im Lexikonautomaten

09.12.2021

Inhalte für heute

1. Tarjan-Tabelle

2. Einfügen und Löschen eines Wortes im Lexikonautomaten

Tarjan-Tabelle

Effizientes Speichern von dürftig besetzten Tabellen (sparse tables)

Grundidee

- Wir speichern die Zustände und Übergänge in einer Liste
- Wir speichern nur tatsächlich existierende Übergänge
- Die Liste hat möglichst wenige Lücken.

$\Sigma = \{a, b, c, d\}$ $Q = \{1, 2, 3, 4, 5, 6\}$

	1	2	3	4	5	6
a	2					3
b						4
c		3				5
d	6	4				

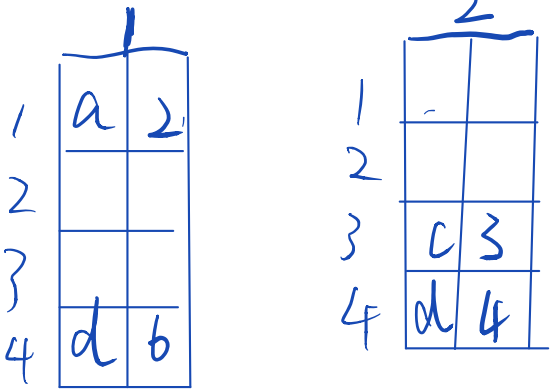
Tarjan-Tabelle

Bausteine

- Ein 'Kamm' in Länge des Alphabets als Eintrag in die Liste
- Tabelle mit 2 Spalten, um Label und Zielzustand zu speichern
- besteht aus **Zellen**, also **Zustandszellen** und **Übergangszellen**

$\Sigma = \{a, b, c, d\}$ $\{a:1, b:2, c:3, d:4\}$

1: $a \to 2$, $d \to b$ 2: $c \to 3$, $d \to 4$



Tarjan-Tabelle

Bausteine

- Ein 'Kamm' in Länge des Alphabets als Eintrag in die Liste
- Tabelle mit 2 Spalten, um Label und Zielzustand zu speichern
- besteht aus **Zellen**, also **Zustandszellen** und **Übergangszellen**

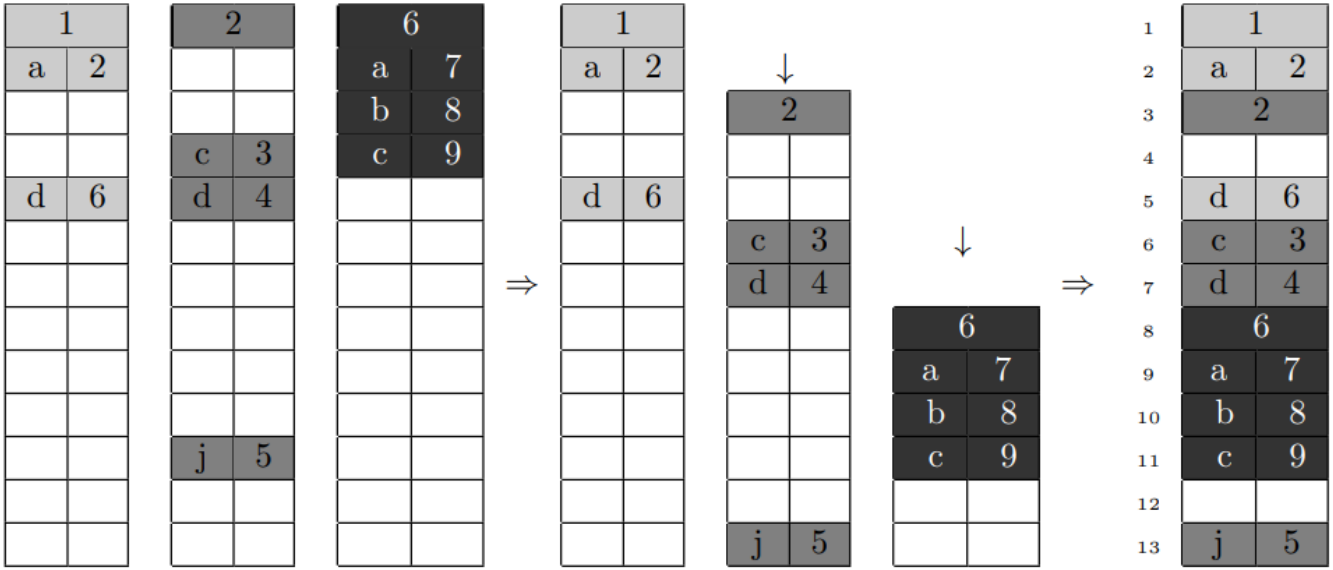


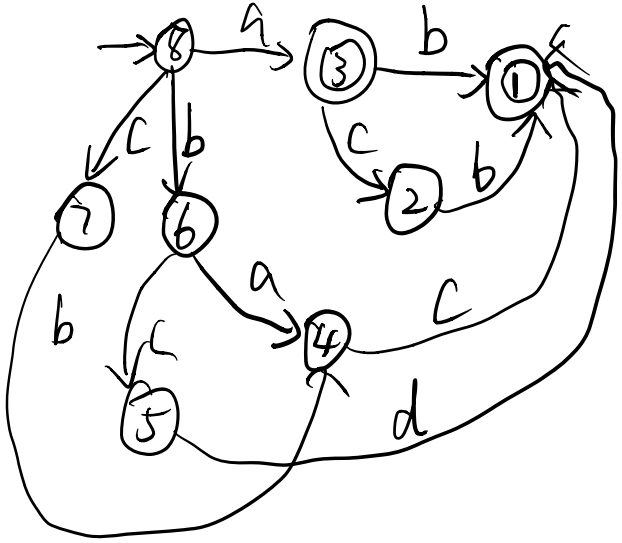
Figure: Quelle Perezautomatenskript

Tarjan-Tabelle - Beispiel

a
ab
acb
bac
bcd
cbc

$\Sigma = \{a, b, c, d\}$

$\{a:1, b:2, c:3, d:4\}$



3	
b	1
c	2

1	1	F
2	2	NF
3	3	F
4	b	1
5	b	1
6	c	2
7	4	NF
8	5	NF
9	7	NF
10	c	1
11	b	1
12	d	1
13	6	NF
14	a	7
15		8
16	c	
17	8	NF
18	a	3
19	b	13
20	c	9
21		

Einfügen und Löschen im Lexikonautomaten

Um aus einer **unsortierten** Liste einen Lexikonautomaten zu bauen.
$\Longrightarrow$ Das entspricht dem wiederholten **Einfügen eines Wortes** in einen Lexikonautomaten.

- Der Automat ist vorher **minimiert**
- Das Wort wird eingefügt oder gelöscht
- Der Automat ist wieder **minimiert**

Einfügen eines Wortes

Wie fügen wir ein neues Wort im sortieten Fall hinzu?

→ Finde das **gemeinsame Präfix** und dessen **Split State**

→ Füge das neue **Suffix** hinzu

→ Minimiere den Pfad

Problem im unsortierten Fall: Es kann mehr als einen eingehenden **Pfad** zum Split State geben

→ Dann würde das Suffix an **mehrere** Präfixe angehängt

→ Dann würde man als Nebeneffekt ungewollt weitere Wörter einfügen.

abc

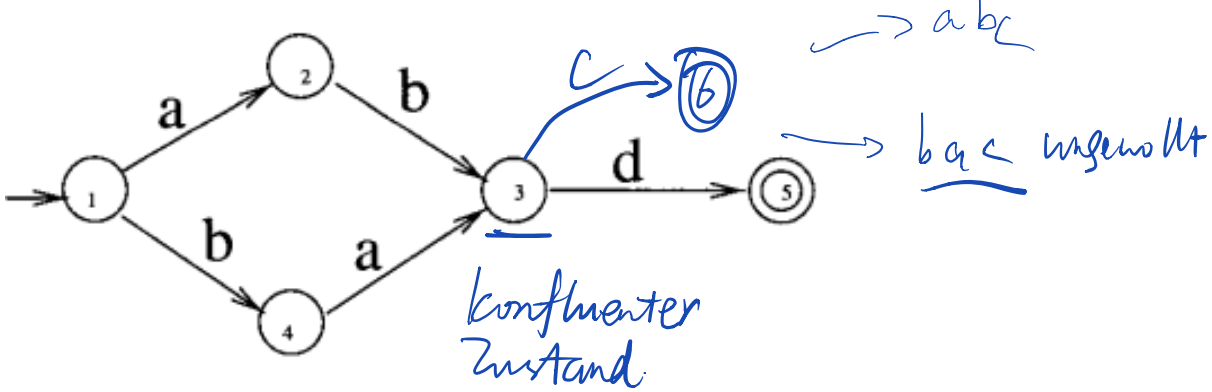


Figure: Quelle Daciuk Paper

Einfügen eines Wortes

Lösung: Es muss der Pfad des Wortes zuerst **isoliert** werden, um die Nebeneffekte zu verhindern.

→ **Konfluente** Zustände auf dem Präfixpfad müssen geklont / gesplittet werden

Konfluente Zustände

- Zustände mit meheren eingehenden Übergängen

Klonen eines konfluenten Zustands

- geklonter Zustand hat den **entsprechend eingehenden** Übergang und **alle ausgehenden** Übergänge des ursprünglichen Zustands

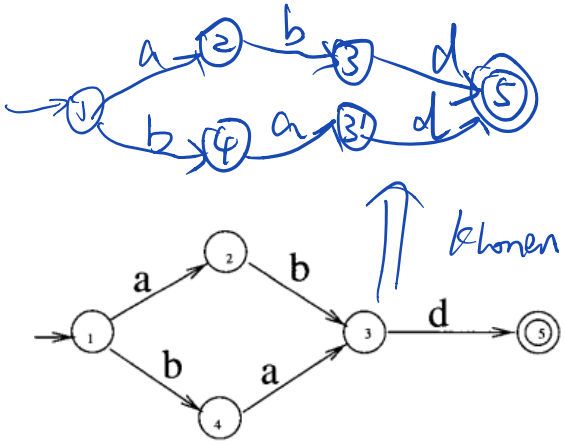


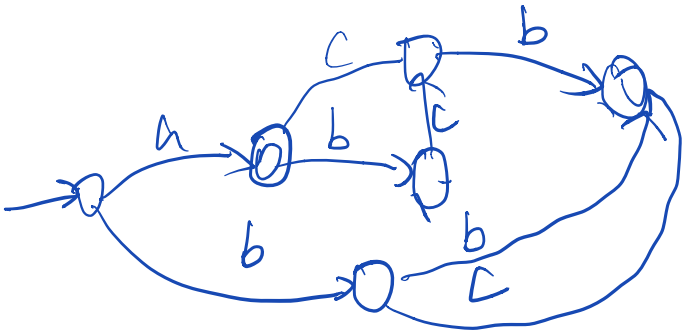
Figure: Quelle Daciuk Paper

Schritte des Einfügens

Einfügen

- Ermittle Split State
- Wenn auf dem Pfad **konfluente** Zustände erscheinen:
  - Klone ab dem ersten konfluenten Zustand **alle** Zustände bis einschließlich **Split State**
- Hänge Suffix an Split State an
- Minimiere

abc
bc
+a
+acb
+bb
+abcb



- abc

Schritte des Löschens

Löschen

- Wenn es auf dem Pfad des zu löschenden Wortes **konfluente** Zustände gibt:
  - Klone ab diesem Zustand **alle** Zustände bis einschließlich **Finalzustand** des Wortes
- Wenn der Finalzustand des zu löschenden Wortes ausgehende Wörter hat:
  - Mache den Finalzustand nicht-final
- Sonst:
  - Lösche die Zustände in Rückwärtsrichtung, bis man auf einem Zustand tritt, der
    - **final** ist und / oder
    - mehr als **einen** ausgehenden Übergang hat
- Minimiere vom erreichten Zustand aus

Weiterer Plan

Nächste Woche wird die Python-Implementierung des Daciuk-Algorithmus besprochen.