Ziel des Projektes ist die Erstellung einer Bibliothek, die den Umgang mit regulären Sprachen ermöglicht.

Projektteilnehmer:
Daniel Dreibrodt, Florian Hemmelgarn, Fabian Ickerott, Sebastian Kowelek, Yacine Smaoui, Konstantin Steinmiller

Anforderungen

Erforderliche Formalismen

- Reguläre Ausdrücke
- Reguläre Grammatiken
- Endliche Automaten

Erforderliche Algorithmen

- Konvertierung
- Minimierung
- Äquivalenzprüfung

Inhaltsverzeichnis

Reguläre Sprache

- Formale Sprache
  - Menge von akzeptablen/korrekten Zeichenketten
  - Zusammengesetzt aus einem Vorrat von Zeichenfolgen (Alphabet)

- Reguläre Sprache
  - definiert durch
    - Regulären Ausdruck
    - Reguläre Grammatik
    - Endlichen Automaten

Regulärer Ausdruck

- Deklarative, algebraische Darstellung regulärer Sprachen
- Einzelne Zeichenkette
- Verknüpfung von Ausdrücken mit Operatoren
  - Aneinanderreihung (Und-Operator): a . b
  - Alternative (Oder-Operator): a | b
  - Wiederholung (Stern-Operator): a*
- Ausdrücke sind beliebige Zeichenfolgen
  - Enthalten keine Leerzeichen & Operatoren

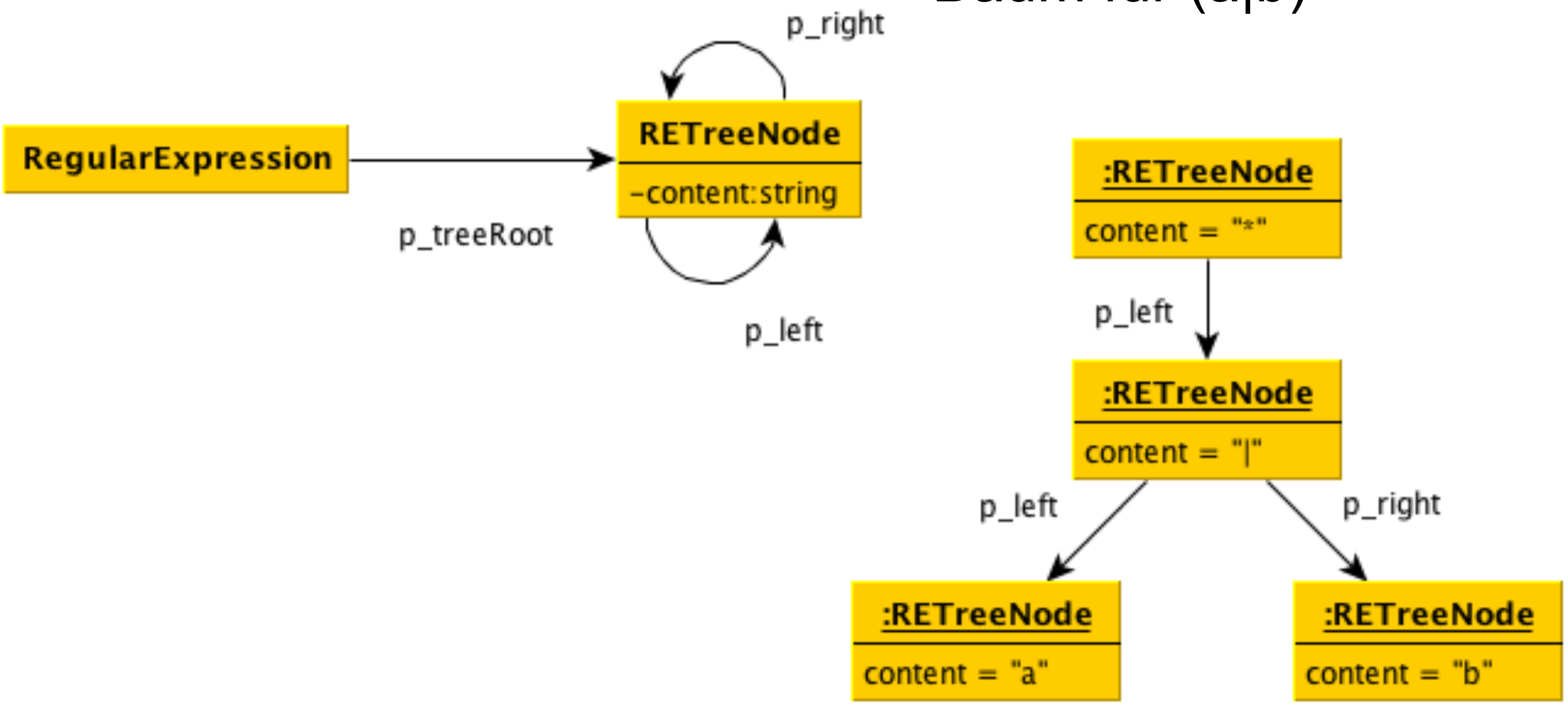
Regulärer Ausdruck - Beispiele

Regulärer Ausdruck	**Akzeptierte Zeichenfolgen**
a.b	“ab”
a|b	“a”, “b”
a.(a|b)	“aa”, “ab”
a*	“”, “a”, “aa”, “aaa”, …
(a|b|c)*	“”, “a”, “ab”, “bca”, “ccba”, …

Regulärer Ausdruck - Datenstruktur

- Speichern als Ausdruckbaum

Baum für (a|b)*



Reguläre Grammatik

Definition:

Eine Grammatik $G = (T, N, P, S)$ besteht aus:

T	Menge aus Terminalsymbolen(kurz: Terminale)
N	Menge einer Nichtterminalsymbole(kurz: Nichtterminale) T und N sind disjunkte Mengen.
$S \in N$	Startsymbol
$P \subseteq N \times V^*$	Menge der Produktionen; $(A, x) \in P$, mit $A \in N$ und $x \in V^*$; statt (A, x) schreibt man $A \rightarrow x$
$V = T \cup N$	heißt auch Vokabular, seine Elemente heißen Symbole

Rechtslineare Grammatik Definition:

Eine Grammatik $G = (T, N, P, S)$ ist eine rechtslineare Grammatik, wenn sie folgenden Anforderungen genügt:

$$X \rightarrow aY$$
$$X \rightarrow a$$
$$X \rightarrow \varepsilon$$

Mit $X, Y \in N$ und $a \in T$.

Reguläre Grammatik - Beispiele

Terminale := {a,b}

Nichtterminale := {S,A,B}

Start-Symbol := S

Produktionen := {

S := a A,

A := b B,

B := b B,

B := b

}

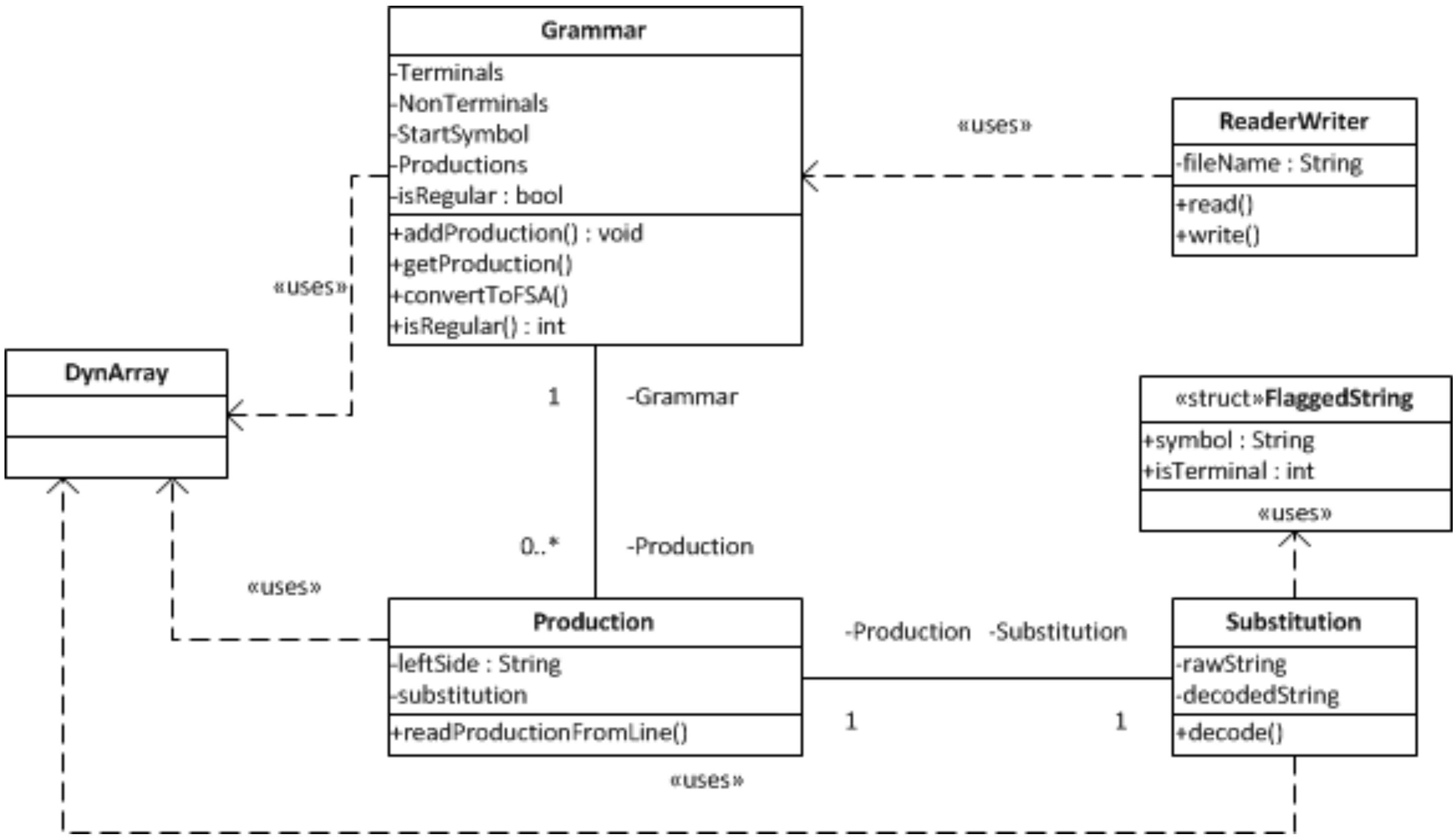
Regulär:

- In den Produktionen sind alle Terminale entweder auf der linken oder rechten Seite der Nichtterminale

Nicht-Regulär:

- In den Produktionen sind Terminale und Nichtterminale gemischt
  - A := a A B b A

Reguläre Grammatik – UML Diagramm



Endlicher Zustandsautomat

Definition:

Ein Automat $A = (Q, \Sigma, \delta, q_0, F)$ besteht aus:

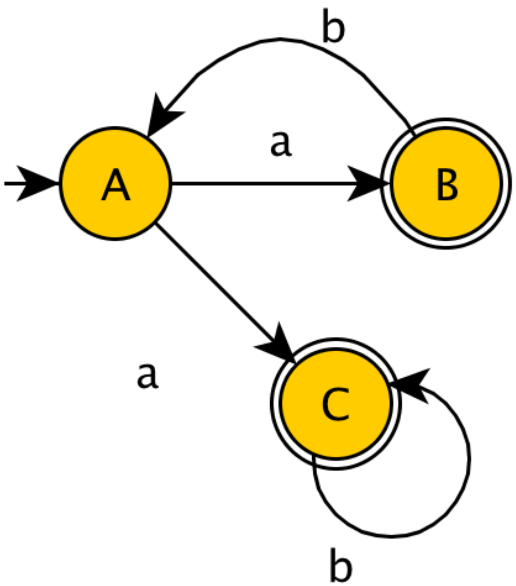
Q	endliche Menge aller Zustände
Σ	Eingabealphabet
$\delta : Q \times \Sigma \rightarrow Q$	Übergangsfunktion
$q_0 \in Q$	Startzustand
$F \subseteq Q$	Endzustände

Deterministischer/ nicht deterministischer Automat

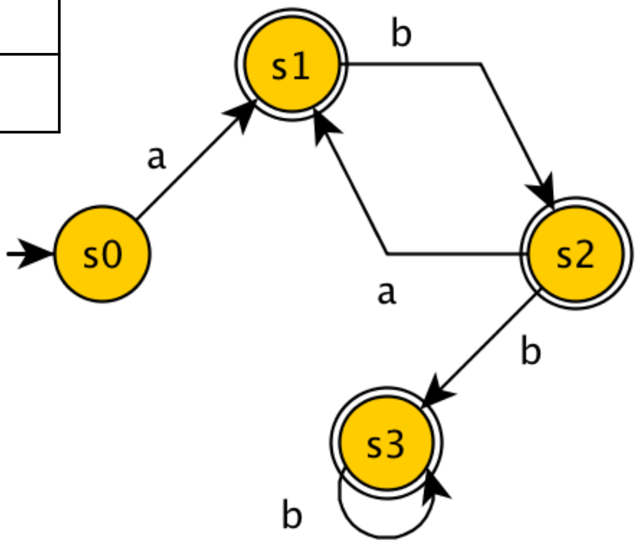
Bei einem deterministischen Automaten ist der Zustandswechsel eindeutig beschrieben.

Endlicher Zustandsautomat – NEA zu DEA

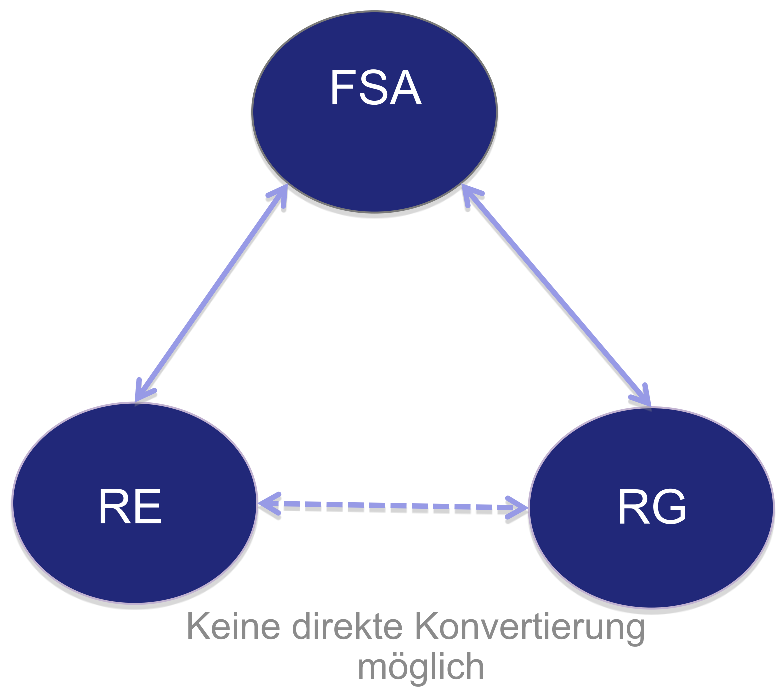
Projektarbeit Automata Tools



	a	**b**
S0{A}	{B,C}	-
S1{B,C}	-	{A,C}
S2{A,C}	{B,C}	{C}
S3{C}	-	{C}

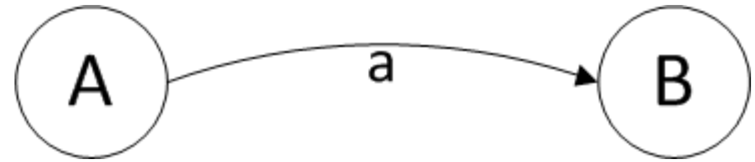


Implementierung: Konvertierungsrichtungen

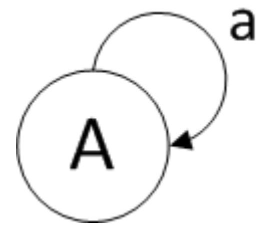


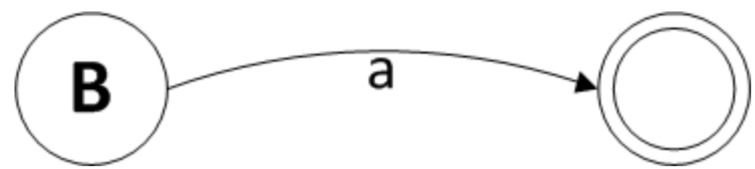
Konvertierung RG ↔ FSA

Konvertiere jede Produktion, in Abhängigkeit von seiner Form

A := a B ⟷ 

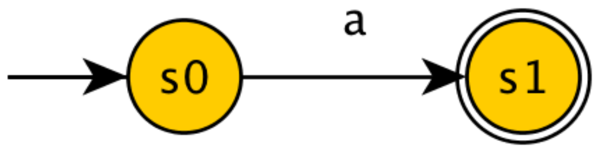
B := <epsilon> ⟷ 

A := a A ⟷ 

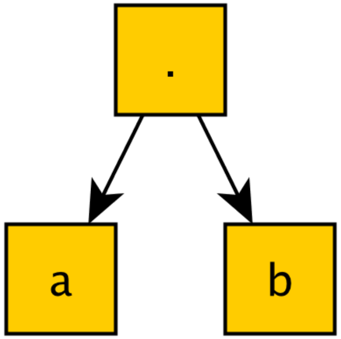
B := a ⟷ 

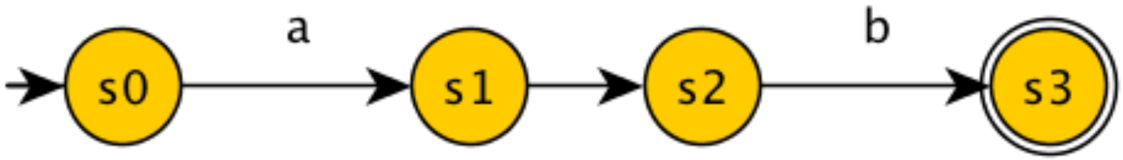
Konvertierung RE zu FSA

- Baumdurchlauf in post-order: Von unten nach oben
- Ausdruck: a



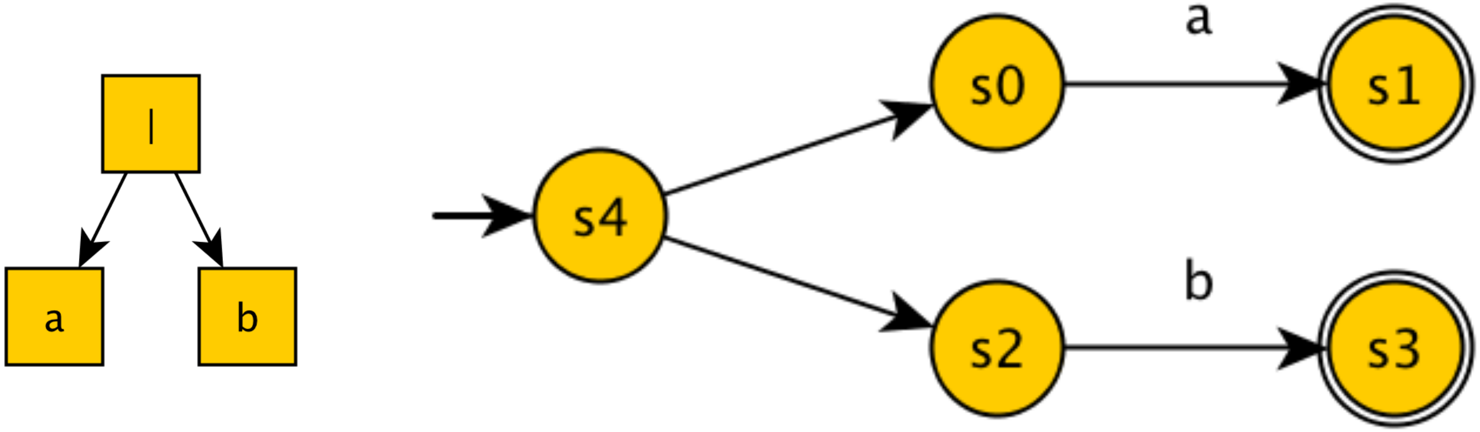
- Und-Verknüpfung: a.b



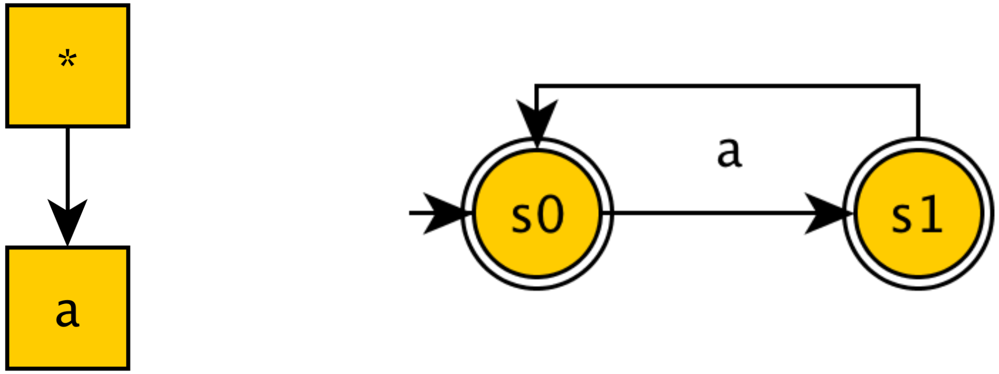


Konvertierung RE zu FSA

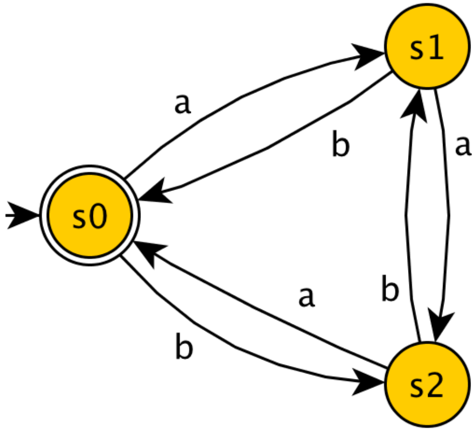
- Oder-Verknüpfung: a|b



- Wiederholung: a*



- Gleichung für die Transitionen aus jeden Zustand
- Löse Gleichungssystem für Startzustand



Gleichungssystem:

s0 = a s1 | b s2 | ε

s1 = a s2 | b s0

s2 = a s0 | b s1

- Ergebnis abhängig von Lösungsreihenfolge
- Implementation: Brzozowskis Algebraische Methode

Minimierung von Finite States Automata

- Ein Minimalautomat für eine Sprache *L* ist derjenige mit der minimalen Anzahl von Zuständen

- Gesucht ist also ein äquivalenter Automat, der eine minimale Anzahl von Zustände hat

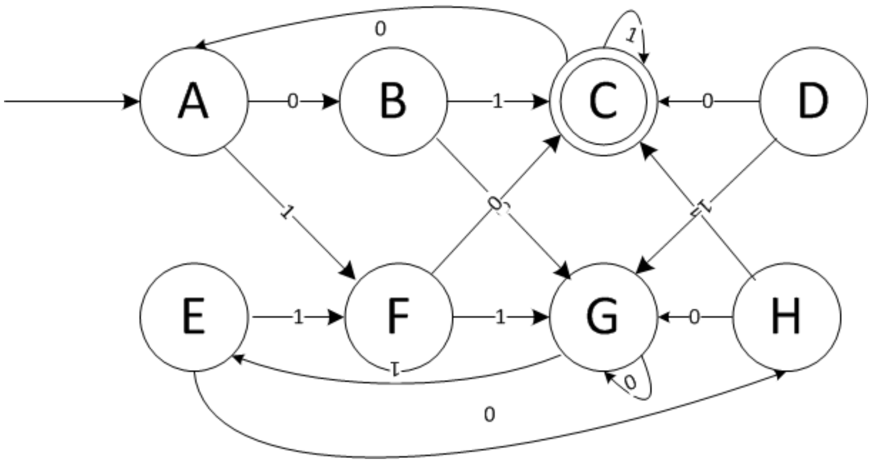
- Zwei Algorithmen sind dafür implementiert:

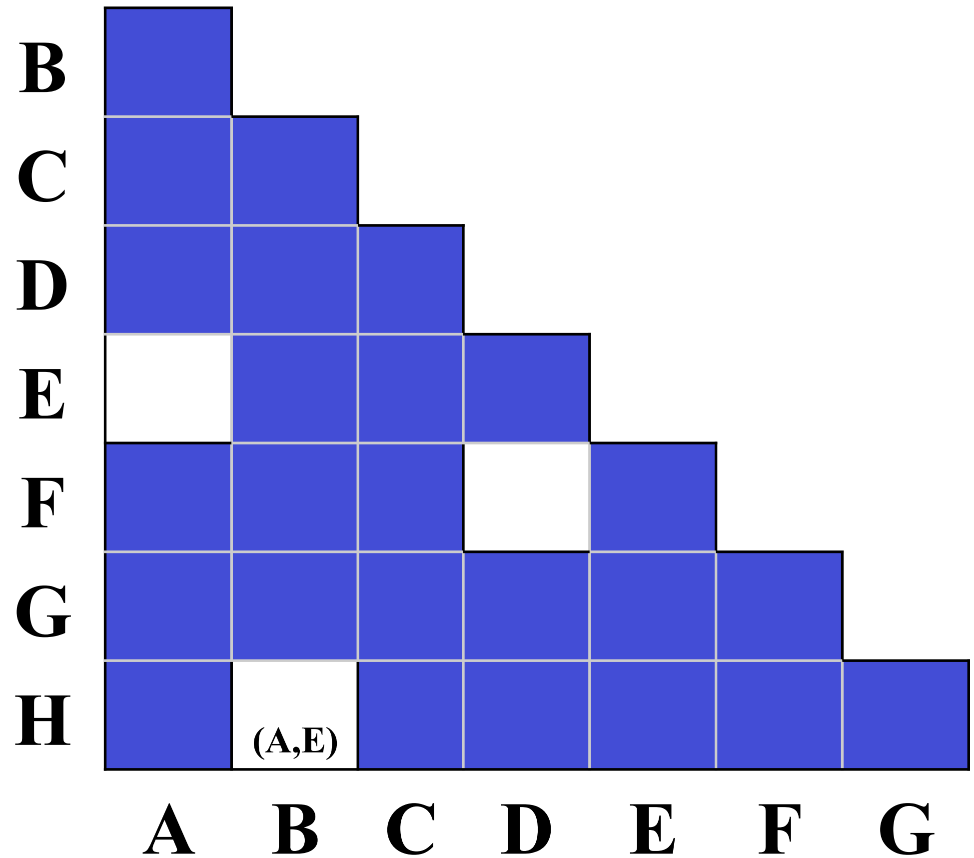
  - Table-filling-Algorithmus: abgeleitet aus dem Äquivalenz-Satz von Myhill und Nerode. In diesem Projekt ist er nur auf „totale“ Automaten anwendbar, kann aber erweitert werden.

  - Moore Algorithmus

Minimierung: Table-filling-Algorithmus

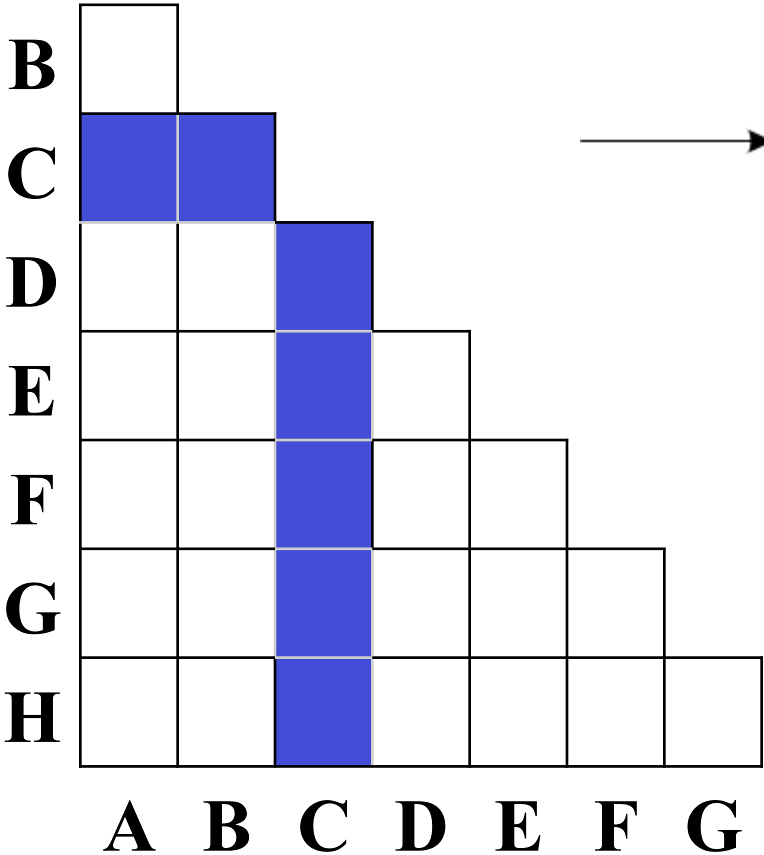
Idee: in einem Matrix, Markiere alle Paare von Zuständen die nicht äquivalent sind.

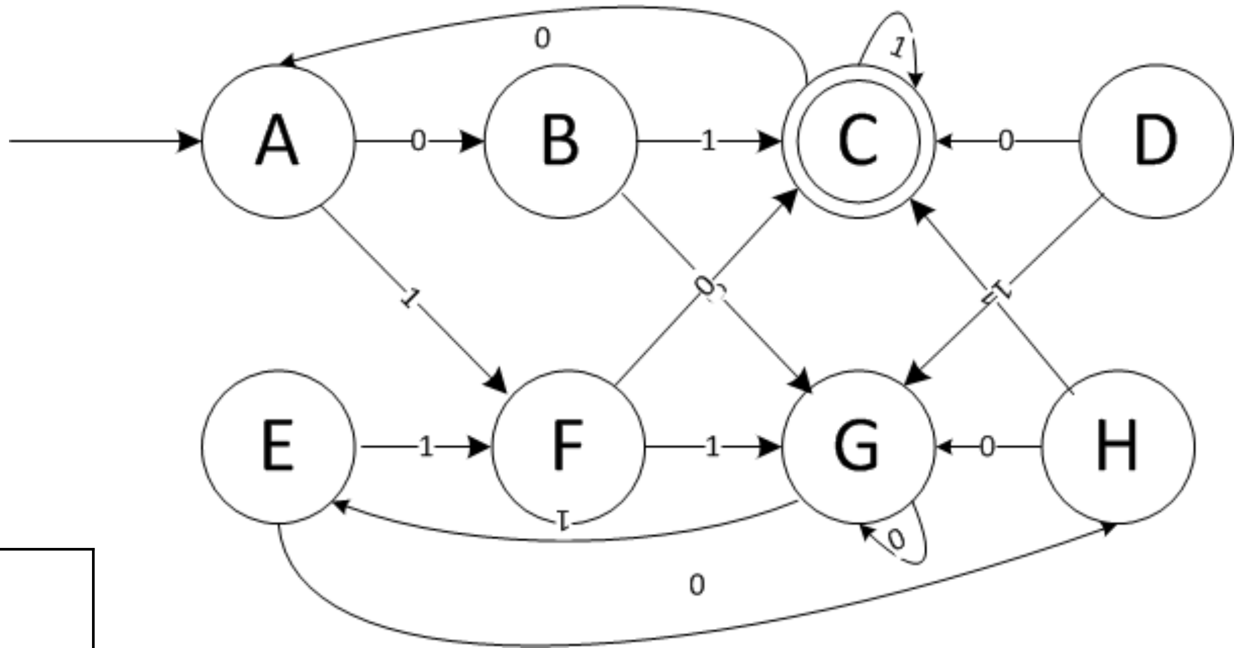




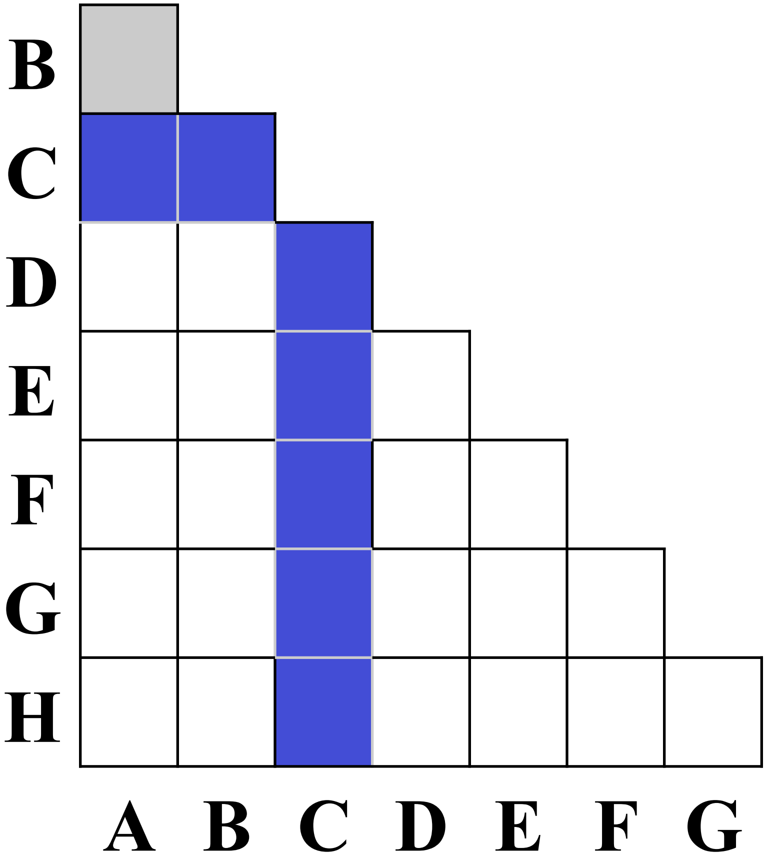
Formal: *zwei Zustände P, Q sind nicht äquivalent, wenn es einen Satz ω gibt, so dass einer von $\delta(P,\omega)$ oder $\delta(Q,\omega)$ ein akzeptierender Zustand und der andere ein nichtakzeptierender Zustand ist.*

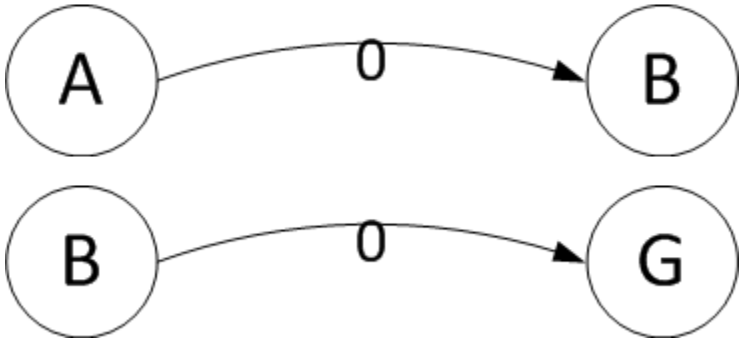
Minimierung: Table-filling-Algorithmus - Beispiel



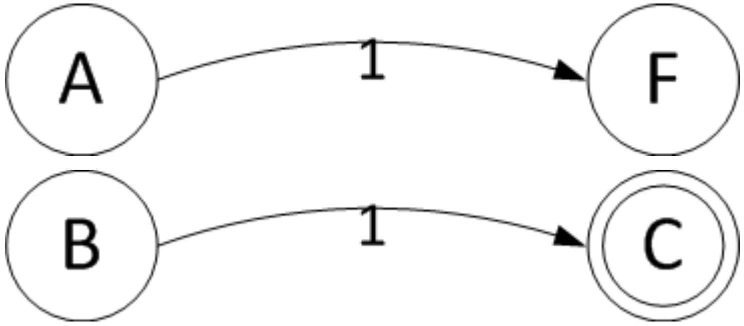


Minimierung: Table-filling-Algorithmus - Beispiel



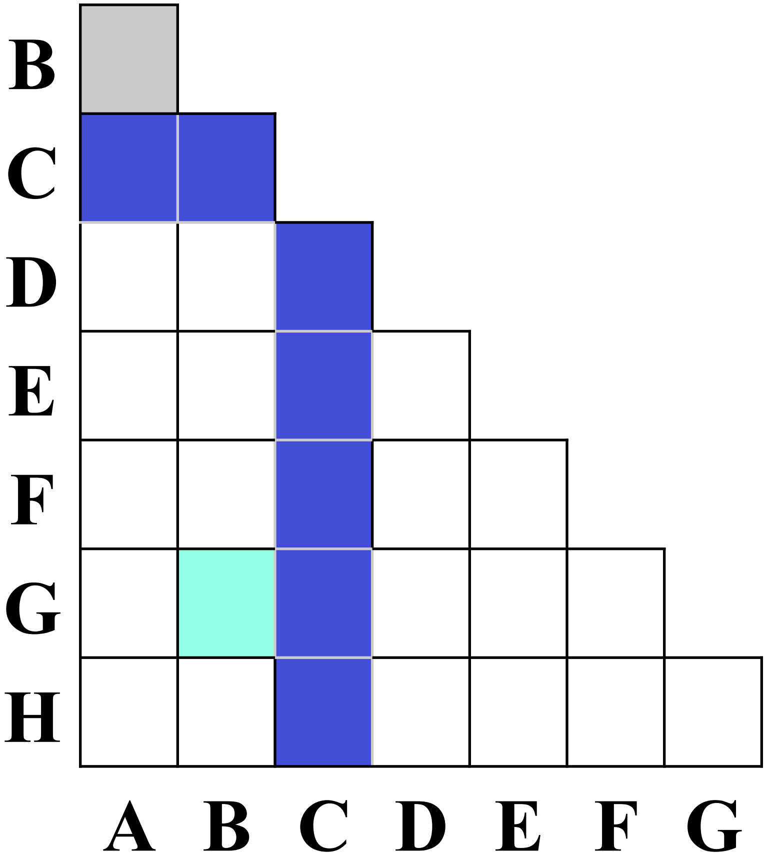


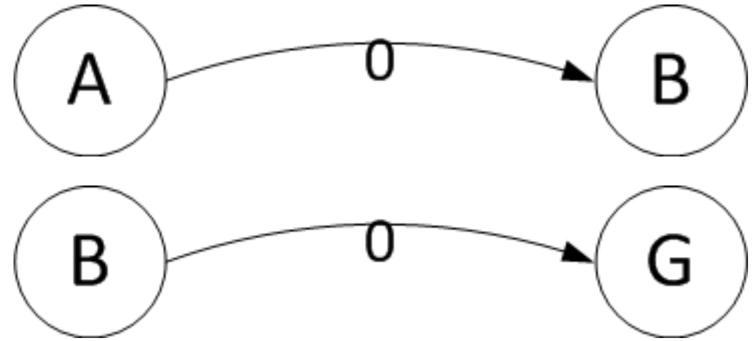
B und G äquivalent?



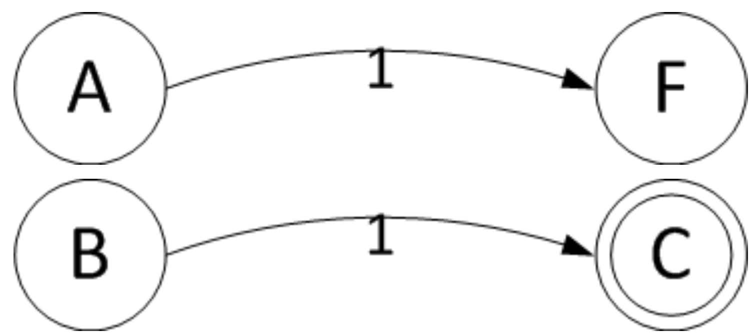
F und C äquivalent?

Minimierung: Table-filling-Algorithmus - Beispiel



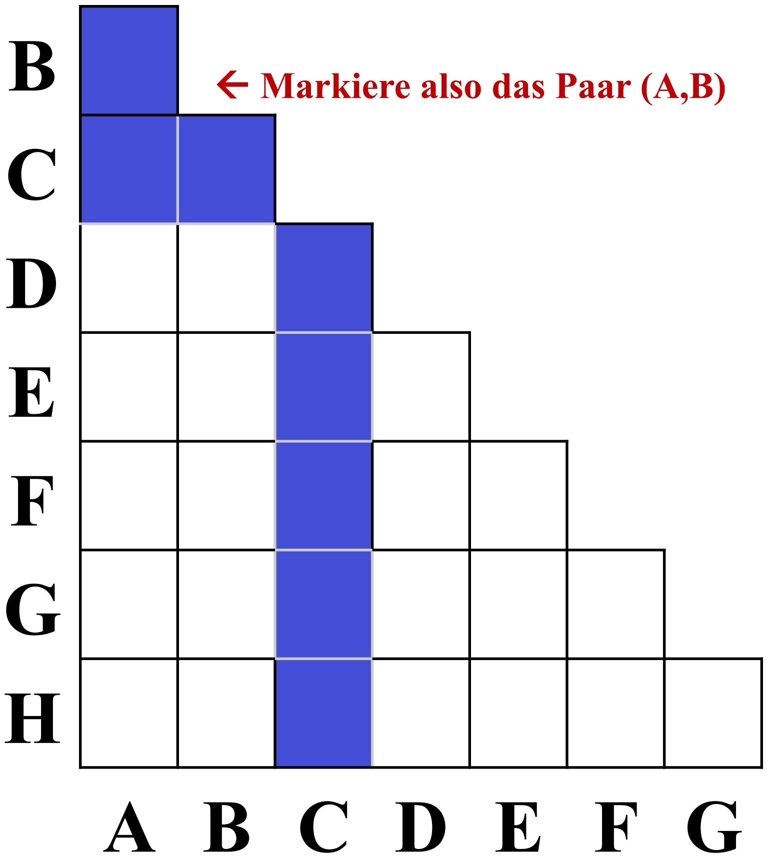


B und G äquivalent?:
(B,G) ist nicht Markiert, also wir wissen noch nicht.

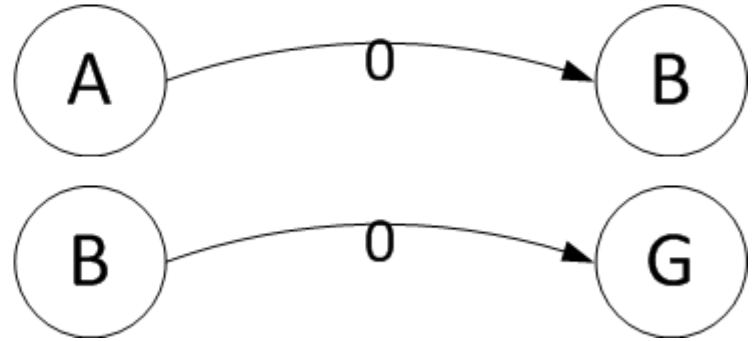


F und C äquivalent?:
Nein !

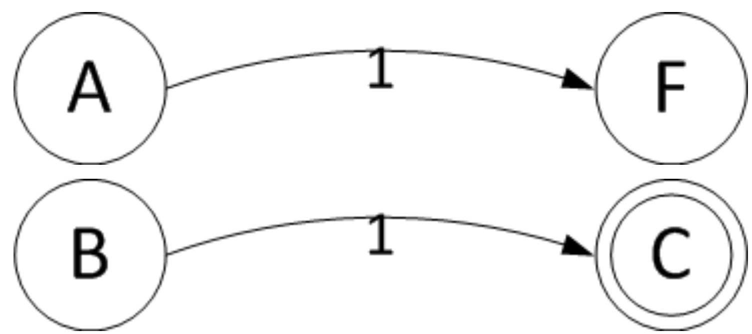
Minimierung: Table-filling-Algorithmus - Beispiel



← Markiere also das Paar (A,B)



B und G äquivalent?:
(B,G) ist nicht Markiert, also wir wissen noch nicht.



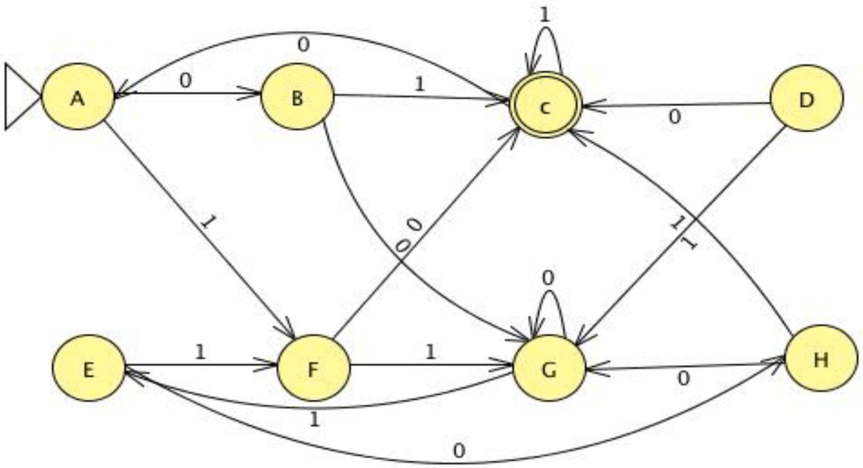
F und C äquivalent?:
Nein !

Minimierung: Table-filling-Algorithmus – Der Algorithmus

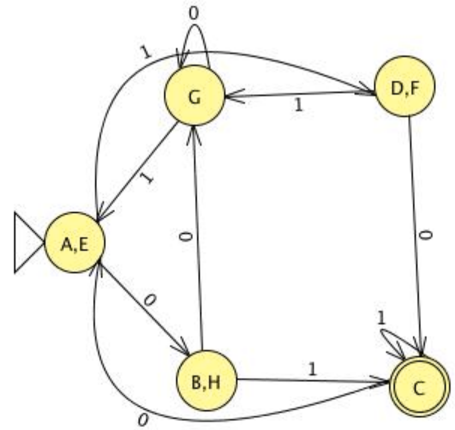
1. Initialisiere alle Paare in der Matrix als nicht-markiert, und ohne “dependencies”.
2. Markiere alle Paare von akzeptierenden Zuständen und nicht akzeptierenden Zuständen.
3. Für alle nicht markierte Paare P,Q und jedes Eingabesymbol a:
   1. Sei $X = \delta(P,a)$, $Y = \delta(Q,a)$.
   2. Falls (X,Y) nicht markiert, füge (P,Q) zu den dependencies von (X,Y) hinzu,
   3. Sonst markiere (P,Q), und markiere alle dependencies von (P,Q).

Minimierung: Moore Algorithmus

Projektarbeit Automata Tools



Beispielautomat



Minimierter Automat

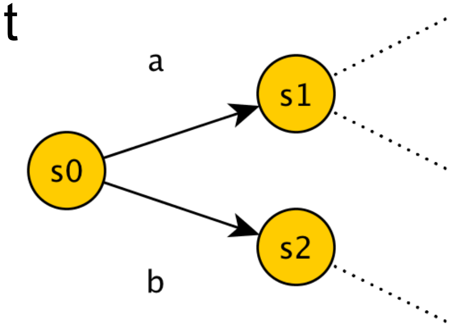
	Rejecting states							Accepting states
	G_0{A	B	D	E	F	G	H}	G_1{C}
0	G_0	G_0	G_0	G_0	G_1	G_0	G_0	G_0
1	G_0	G_1	G_0	G_0	G_0	G_0	G_1	G_1

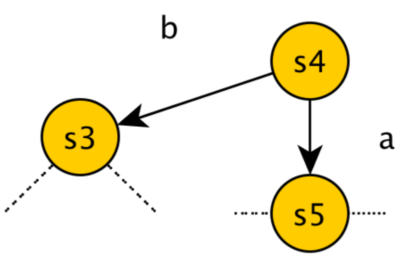
	G_0{A	D	E	G}	G_1{B	H}	G_2{F}	G_3{C}
0	G_1	G_0	G_1	G_0	G_0	G_0	G_3	G_0
1	G_2	G_0	G_2	G_0	G_3	G_3	G_0	G_3

- •Unterteile states in accepting and rejecting und behandle im Folgenden beide Bereiche unabhängig von einander
- •Gib für jeden Übergang die Endgruppe jedes states an
- •Erzeuge für jede Abweichung eine neue Gruppe und fülle sie mit den entsprechenden states
- •Wiederholung bis keine Abweichung vorhanden → Automat ist minimal

Äquivalenz

- Eingabe: Zwei deterministische, minimale Automaten
- Bei unteschiedlicher Zustandsanzahl
  - Trivialer Fall: Automaten sind nicht äquivalent
- Ansonsten
  - Überprüfung ob Startzustände der Automaten äquivalent sind
- Zwei Zustände sind äquivalent, wenn
  - Für alle Elemente des Alphabets die jeweils erreichbaren Zustände äquivalent sind





Vielen Dank für Ihre Aufmerksamkeit