Algorithmen und Datenstrukturen

Theoretische Grundlagen der Informatik

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

4. Juni 2024

Algorithmen und Datenstrukturen

Minimierung von deterministischen endlichen Automaten

Prof. Dr. rer. nat. Jörg Striegnitz

Fachhochschule Aachen
striegnitz@fh-aachen.de

4. Juni 2024

Motivation 1

- Zu einer Sprache L gibt es grundsätzlich **mehrere** Automaten M mit $L(M) = L$

  Diese unterscheiden sich ggf. in Zahl der Zustände

- Es gibt aber einen, abgesehen von der Bezeichnung der Zustände, **eindeutigen, minimalen** Automaten!

  und es gibt effiziente Algorithmen diesen zu bestimmen

- **Damit**: Beweis der Äquivalenz zweier Automaten M_1 und M_2 trivial!

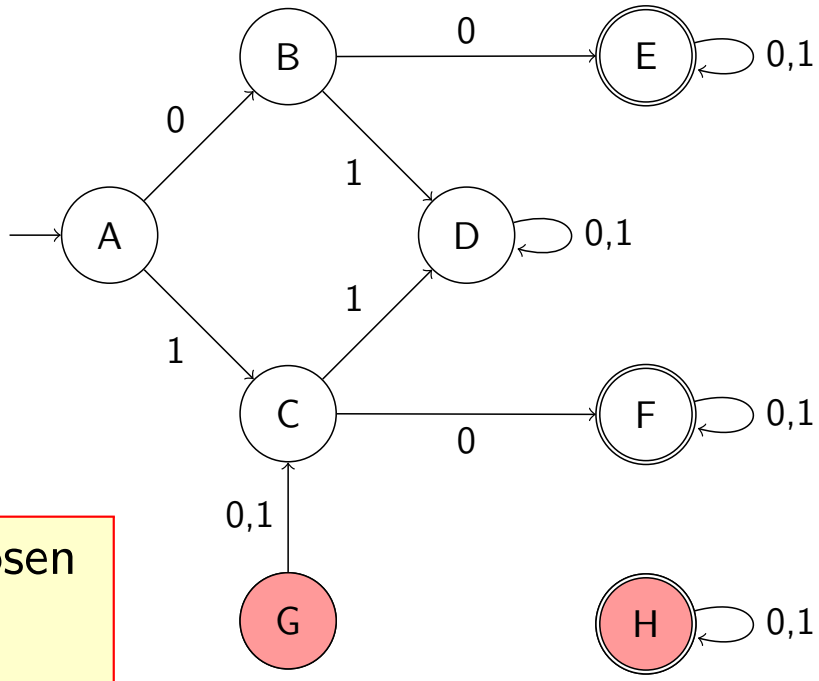
  Minimiere beide und vergleiche die Ergebnisse ...

  Anmerkung: $M_1 \equiv M_2 \quad \Leftrightarrow \quad L(M_1) = L(M_2)$

Motivation 2

Minimieren eines Automaten bedeutet

1. **Unerreichbare Zustände** streichen
2. **Äquivalente Zustände** zusammenfassen



Wir setzen voraus, dass wir 1. effizient lösen können

Algorithmen dazu werden wir noch kennenlernen

Aufgabe: Finde effizienten Algorithmus für Zusammenfassung äquivalenter Zustände.

Äquivalente Zustände eines DEA

Definition 1.43 (Äquivalente Zustände eines DEA)

Sei $M = (Q, \Sigma, \delta, q_0, F)$ ein deterministischer endlicher Automat. Zwei Zustände $p \in Q$ und $q \in Q$ sind genau dann **äquivalent** (in Zeichen: $p \equiv q$), wenn

$$\forall w \in \Sigma^* \quad : \quad \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$$

In Σ^* sind unendlich viele Wörter enthalten

- Bedingung der Definition bietet damit keine unmittelbare Grundlage für einen Algorithmus

Grundsätzliches Vorgehen

1. Äquivalenztest: ermittle für alle Zustandspaare $(p, q) \in Q \times Q$, ob $p \equiv q$
2. bilde anschießend Äquivalenzklassen

   d.h. berücksichtige transitive Eigenschaften: ist z.B. $p \equiv q$ und $q \equiv r$, so auch $p \equiv r$ und somit gehören p, q, r zu einer Äquivalenzklasse
3. konstruiere neuen Automaten

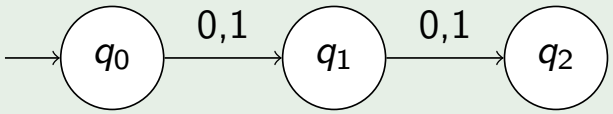
Problem: für den Äquivalenztest müssen wir gem. Definition 1.43 unendlich viele Wörter aus Σ^* betrachten:

$$\forall w \in \Sigma^* \quad : \quad \hat{\delta}(p, w) \in F \iff \hat{\delta}(q, w) \in F$$

1. Versuch - Idee zur Vereinfachung

Beobachtung 1.1 (Zyklen bei Berechnung)

Seien $M = (Q, \Sigma, \delta, q_0, F)$ ein DEA und $w \in \Sigma^*$ mit $|w| = |Q|$ ein Wort, dessen Länge der Zahl der Zustände entspricht. Dann wird bei einer Berechnung für w mindestens ein Zustand doppelt besucht.
Beispiel:



Egal wie Sie die Transitionfunktion am Zustand q_2 fortsetzen, mit Lesen des dritten Buchstabens wird ein Zustand wiederholt besucht.

Nehmen wir an, uns würde eine gute Idee kommen, diese Bedingung irgendwie ausnutzen zu können, so dass es ausreichend wäre, sich beim Äquivalenztest auf Wörter der Länge $< |Q|$ zu beschränken.

Denken Sie an dieser Stelle bitte nicht weiter darüber nach, wie das gehen könnte - wir werden gleich sehen, dass dieses Vorgehen ohnehin zu ineffizient ist ....

1. Versuch - Beurteilung der Effizienz (1)

Der vereinfachte Äquivalenztest wäre

... ohne wirklich zu wissen, wie wir damit einen Algorithmus basteln ...

$$\forall w \in \Sigma^* \text{ mit } |w| < |Q| \;:\; \hat{\delta}(p, w) \in F \;\Leftrightarrow\; \hat{\delta}(q, w) \in F$$

Wie viele Tests müssen wir durchführen?

Sei $n = |Q|$; betrachten wir den Fall $\Sigma = \Sigma_{Bool}$: es gibt

- $\binom{n}{2}$ viele Zustandspaarungen (**Beachte:** $\equiv$ ist reflexiv und symmetrisch!) und
- 2^n viele Wörter der Länge n

Damit erhalten wir

$$T(n) = \overbrace{\binom{n}{2}}^{\#\text{ Paare}} \cdot \underbrace{\sum_{i=0}^{n-1} 2^i}_{\#\text{ Wörter}}$$

Diese Funktion wächst rasant (exponentiell)!

1. Versuch - Beurteilung der Effizienz (2)

$$
\begin{aligned}
T(n) &= \binom{n}{2} \cdot \sum_{i=0}^{n-1} 2^i \\
&= \frac{n!}{(n-2)!2!} \cdot (2^n - 1) \\
&= \frac{n \cdot (n-1)}{2} \cdot (2^n - 1) \\
&= n \cdot (n-1) \cdot (2^{n-1} - \frac{1}{2}) \\
&= (n^2 - n) \cdot (2^{n-1} - \frac{1}{2})
\end{aligned}
$$

n	$T(n)$
10	46.035
11	112.585
12	270.270
13	638.898
14	1.490.853
⋮	⋮
20	199.229.250
⋮	⋮
30	467.077.693.005

! Ansatz für Automaten mit höherer Zustandszahl ungeeignet!

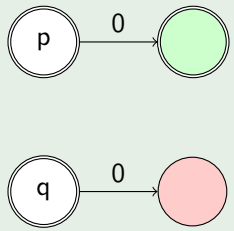
Eigenschaften von $\equiv$ (1)

Gehen wir davon aus es gilt $p \equiv q$ - was können wir daraus ableiten?

Beobachtung 1.2

$$p \equiv q \quad \Rightarrow \quad p \in F \Leftrightarrow q \in F$$

Die Umkehrung gilt im allgemeinen nicht:



p und q sind zwar beide akzeptierend, können aber nicht äquivalent sein!

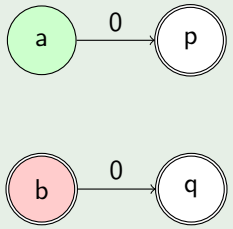
Eigenschaften von $\equiv$ (2)

Gehen wir davon aus es gilt $p \equiv q$ - was können wir daraus ableiten?

Beobachtung 1.3

$$p \equiv q \quad \Rightarrow \quad \forall a \in \Sigma : \delta(p, a) \equiv \delta(q, a)$$

Die Umkehrung gilt im allgemeinen nicht:



p und q seien äquivalent - a und b sind es definitiv nicht!

Übergang zu einem analogen Problem

Negieren wir die Aussagen, können wir auf **Nicht-Äquivalenz** schließen:

Nicht-Äquivalenz von Zustandspaaren

$$p \in F \nLeftrightarrow q \in F \quad \Rightarrow \quad p \not\equiv q \tag{1}$$

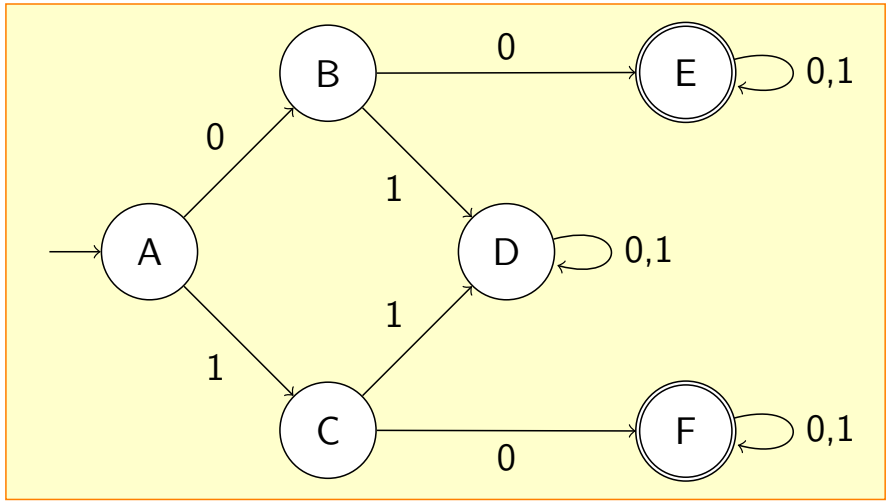
$$\exists a \in \Sigma : \delta(p, a) \not\equiv \delta(q, a) \quad \Rightarrow \quad p \not\equiv q \tag{2}$$

Wir entwickeln mit diesen Regeln einen **Abhängigkeitegraphen**
(engl: *dependency graph*)

- **Knoten**: Zustandspaare aus $Q \times Q$
  wegen Symmetrie und Reflexivität von $\equiv$ nicht alle Paare!
- **Kanten**: Abhängigkeiten; $(q, r) \to (s, t)$ bedeutet, dass $q \not\equiv r$, falls $s \not\equiv t$.

und verwenden ihn, um für alle Zustandspaare die Nicht-Äquivalenz zu entscheiden.

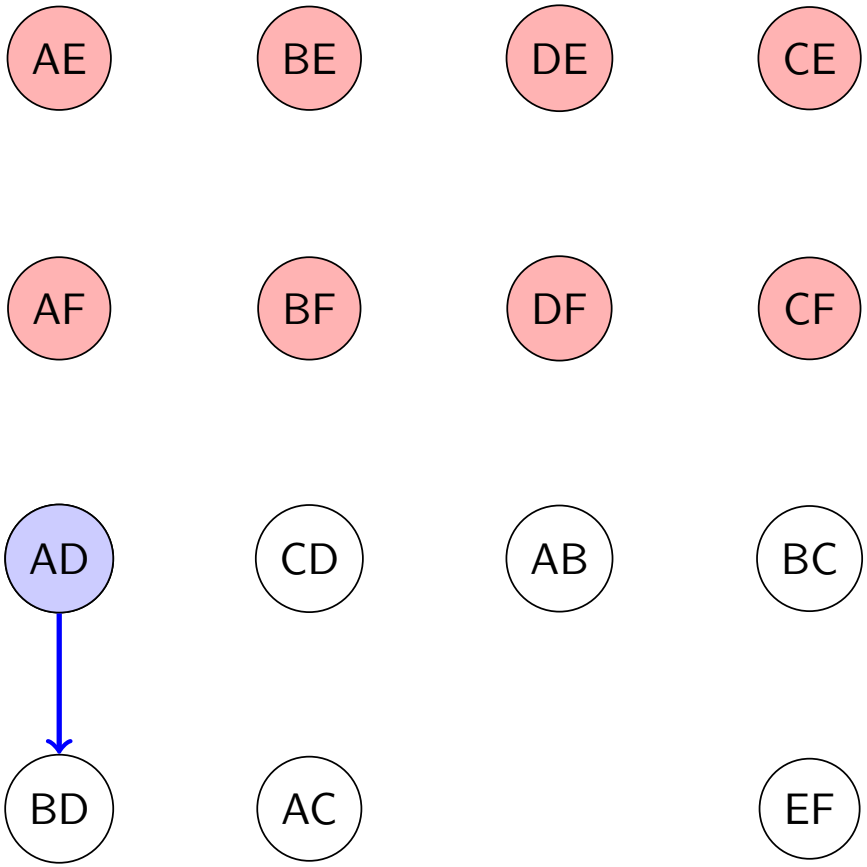
2. Versuch: Konstruktion des Abhängigkeitsgraphen (1)

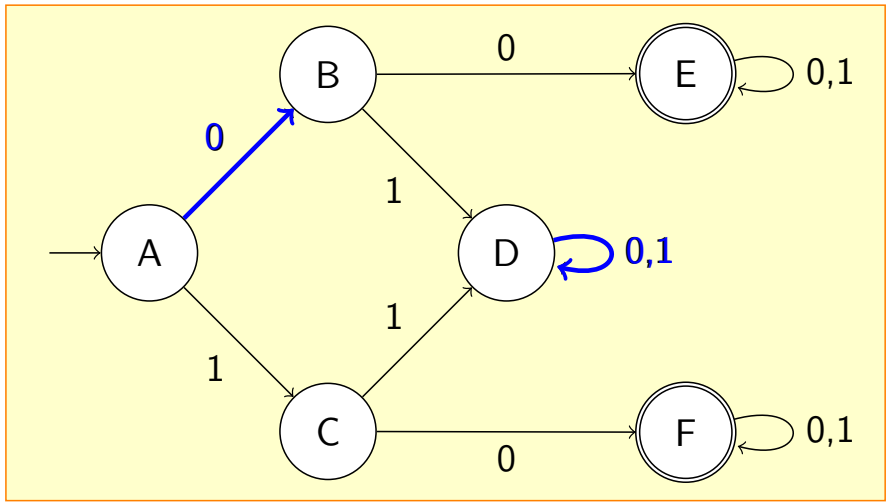


Knoten, deren Zustandspaar nicht äquivalent sein kann, markieren wir rot.

Anwenden der ersten Regel: $p \in F \nLeftrightarrow q \in F \Rightarrow p \not\equiv q$

2. Versuch: Konstruktion des Abhängigkeitsgraphen (2)



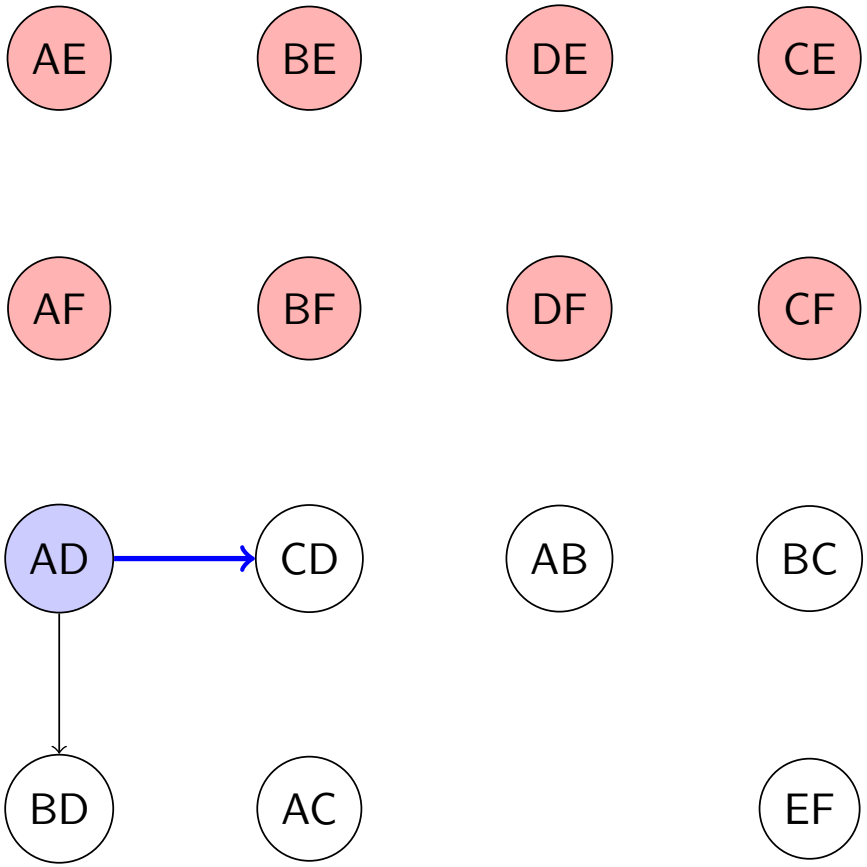


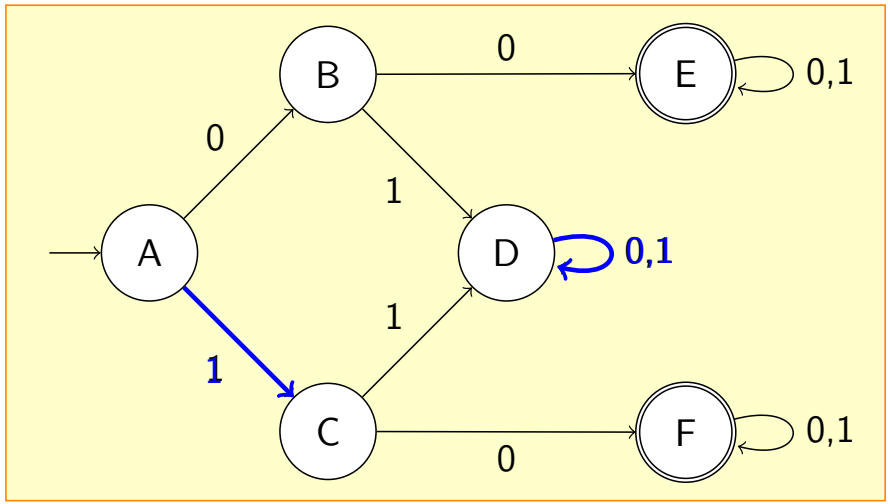
Für die anderen Knoten wenden wir die 2. Regel an

$\exists a \in \Sigma : \delta(p, a) \not\equiv \delta(q, a) \Rightarrow p \not\equiv q$

A und D sind nicht äquivalent, wenn B und D nicht äquivalent sind!

2. Versuch: Konstruktion des Abhängigkeitsgraphen (3)



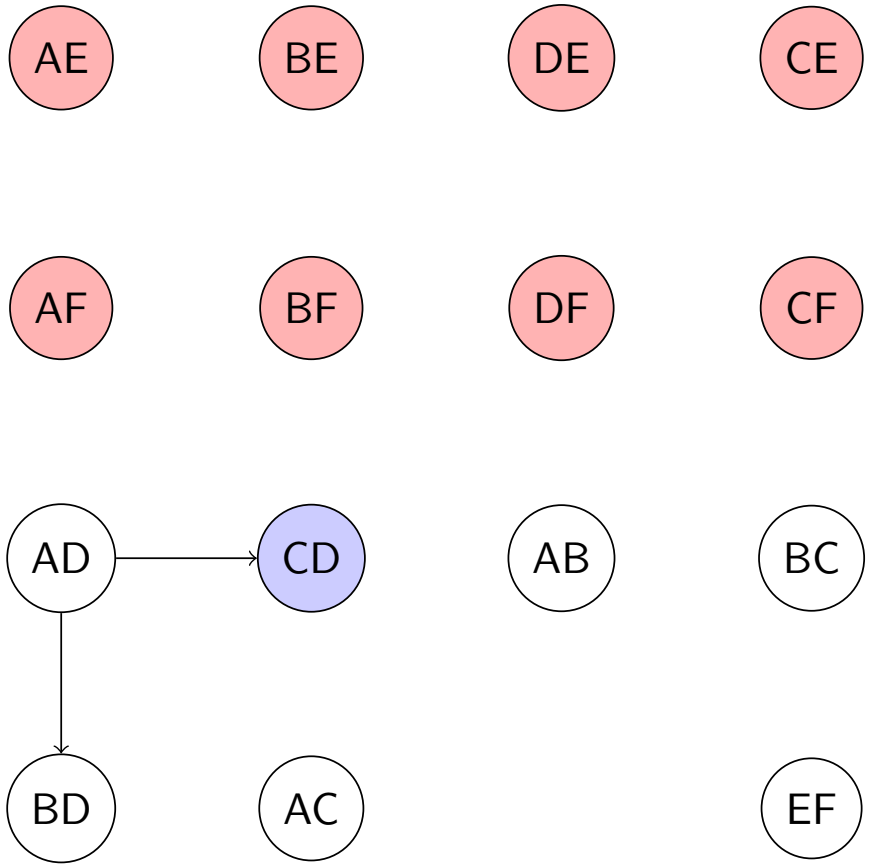


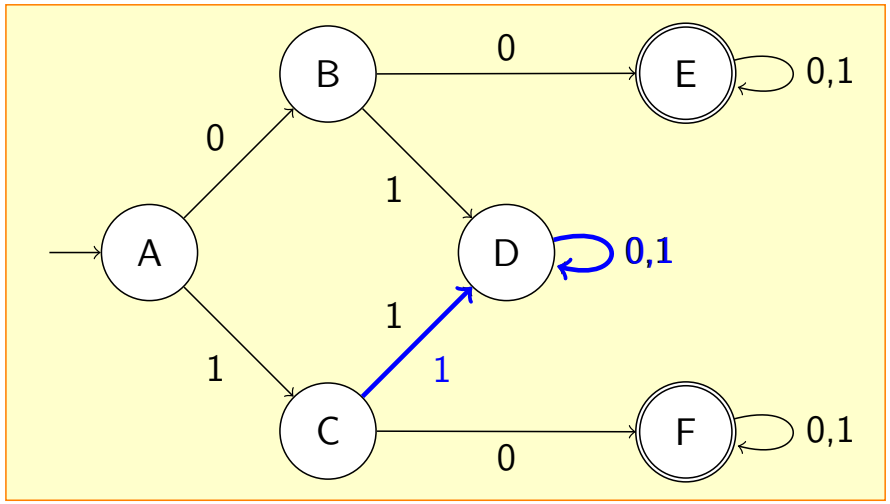
Für die anderen Knoten wenden wir die 2. Regel an

$\exists a \in \Sigma : \delta(p, a) \notin \delta(q, a) \Rightarrow p \notin q$

A und D sind nicht äquivalent, wenn C und D nicht äquivalent sind!

2. Versuch: Konstruktion des Abhängigkeitsgraphen (4)





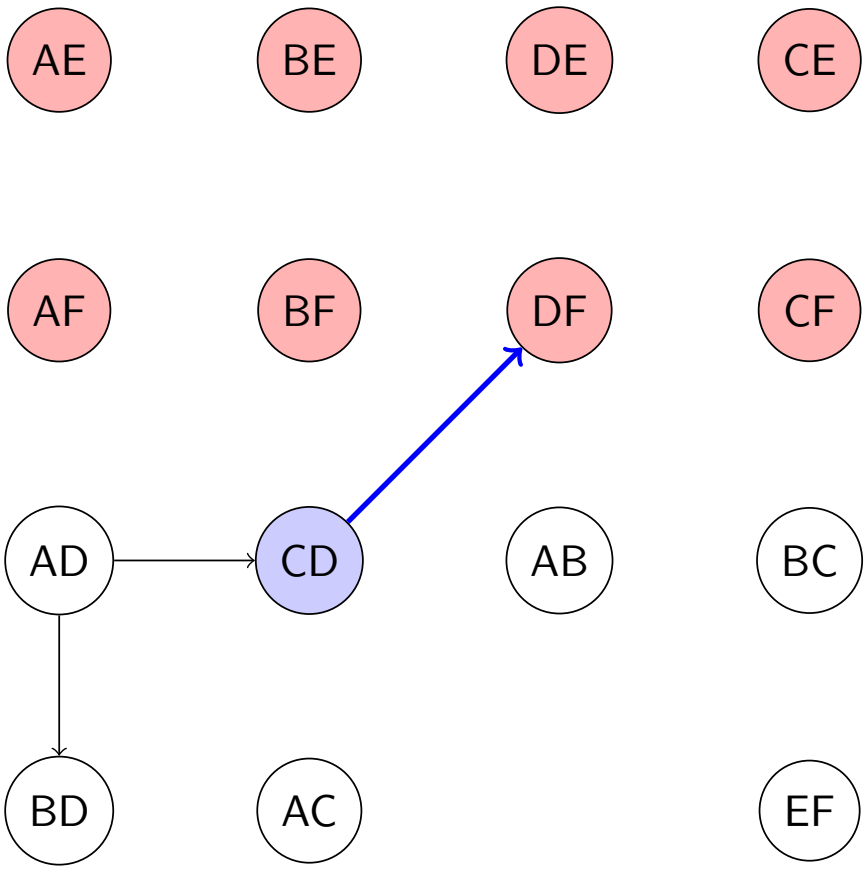
Für die anderen Knoten wenden wir die 2. Regel an

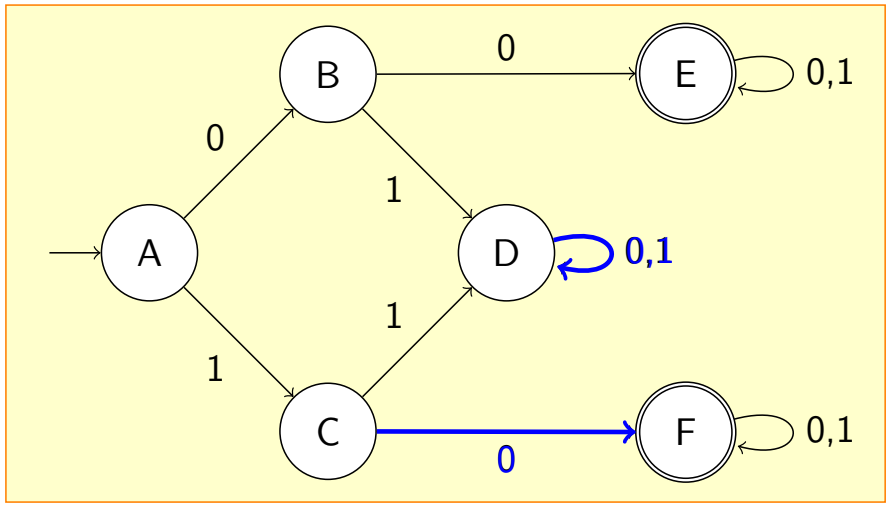
$\exists a \in \Sigma : \delta(p,a) \not\equiv \delta(q,a) \Rightarrow p \not\equiv q$

C und D sind nicht äquivalent, wenn D und D nicht äquivalent sind!

Da $D \equiv D$, hilft uns das nicht weiter ....

2. Versuch: Konstruktion des Abhängigkeitsgraphen (5)





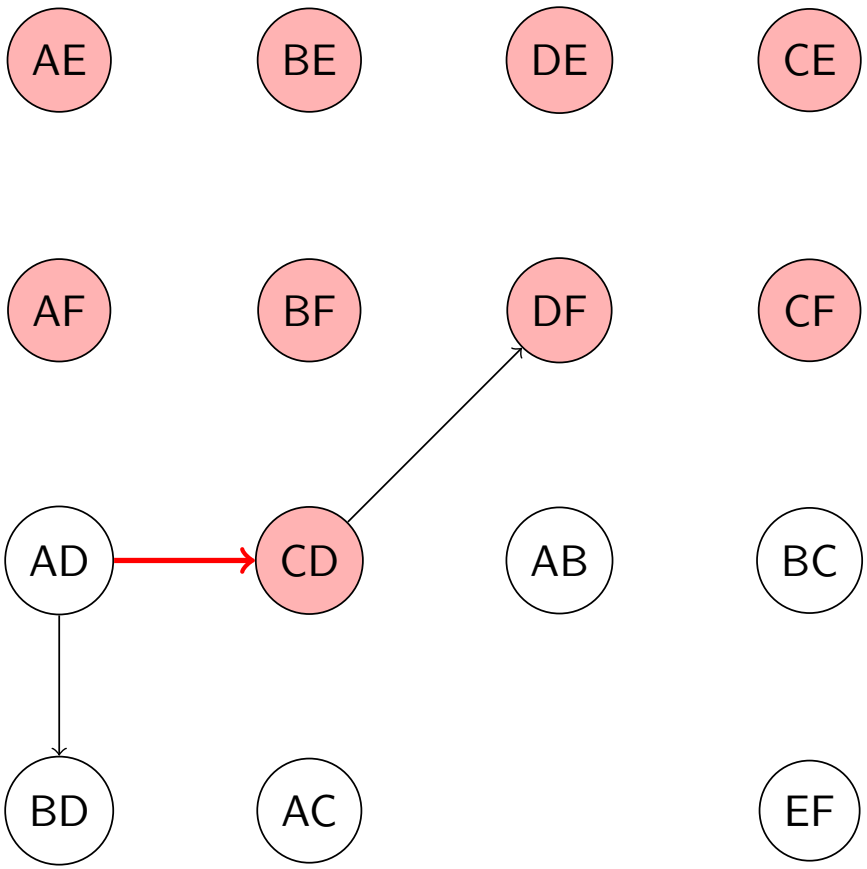
Für die anderen Knoten wenden wir die 2. Regel an

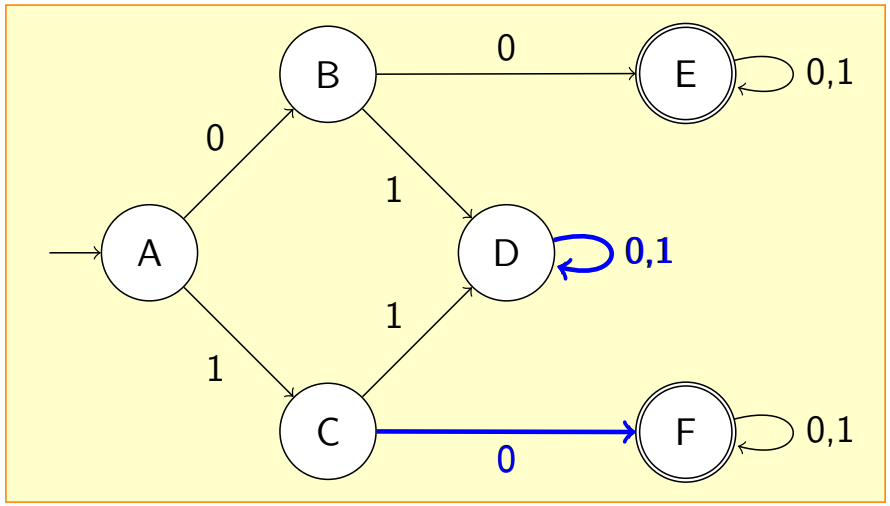
$\exists a \in \Sigma : \delta(p,a) \not\equiv \delta(q,a) \Rightarrow p \not\equiv q$

C und *D* sind nicht äquivalent, wenn *D* und *F* nicht äquivalent sind!

was der Fall ist - wir können *CD* also ebenfalls rot markieren!

2. Versuch: Konstruktion des Abhängigkeitsgraphen (6)





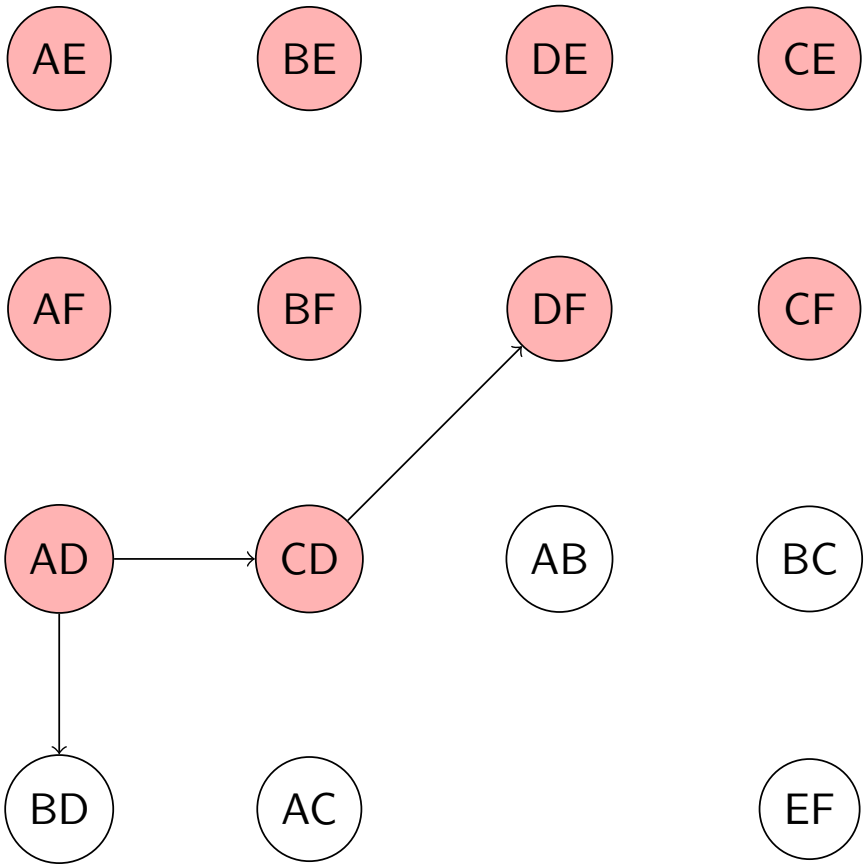
Für die anderen Knoten wenden wir die 2. Regel an

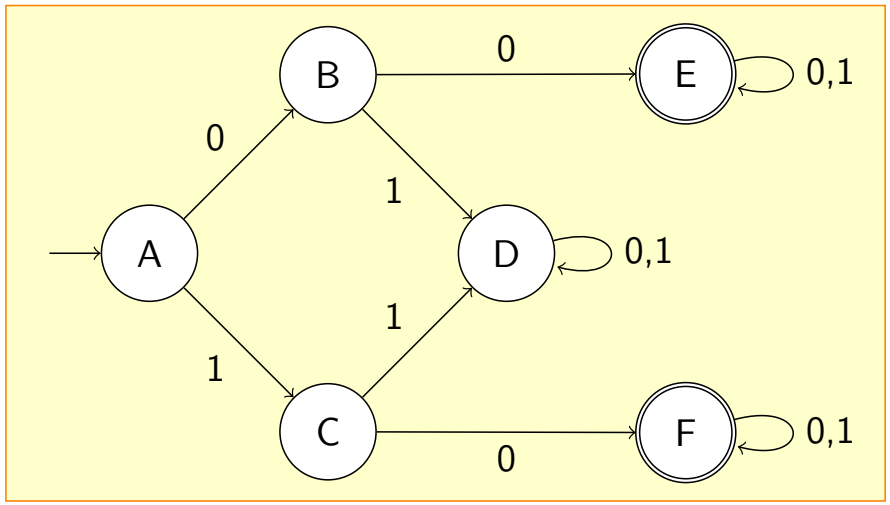
$\exists a \in \Sigma : \delta(p,a) \notin \delta(q,a) \Rightarrow p \notin q$

A und *D* sind nicht äquivalent, wenn *C* und *D* nicht äquivalent sind!

was der Fall ist - wir können *AD* also ebenfalls rot markieren!

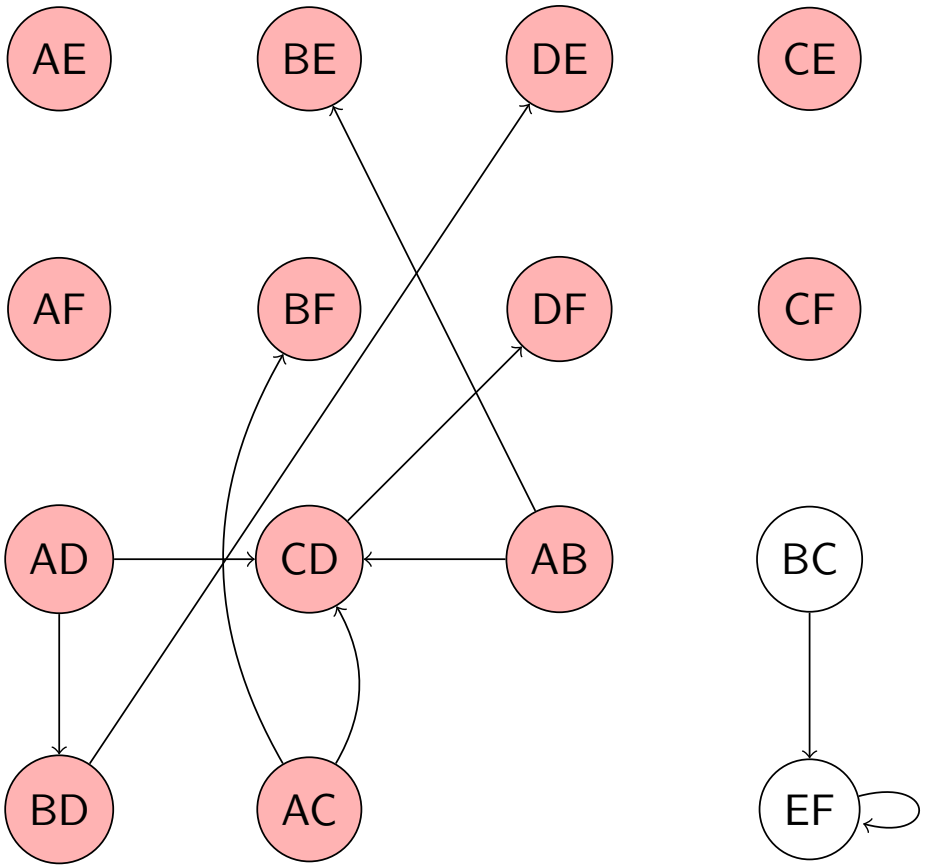
2. Versuch: Konstruktion des Abhängigkeitsgraphen (7)

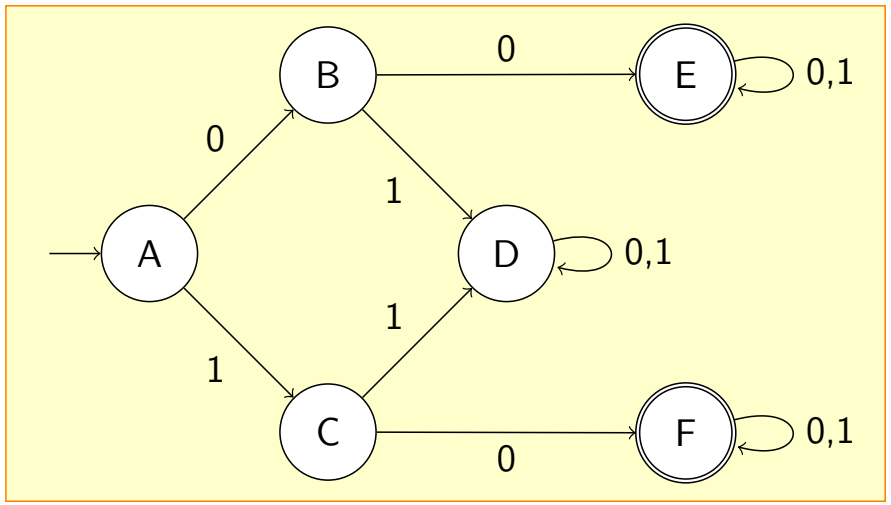




Setzen wir dieses Verfahren fort, gelangen wir zu ...

2. Versuch: Konstruktion des Abhängigkeitsgraphen (8)





... und erhalten $B \equiv C$ und $E \equiv F$

2. Versuch - Beurteilung der Effizienz

Wie viele Tests müssen wir durchführen?

Sei $n = |Q|$; betrachten wir den Fall $\Sigma = \Sigma_{Bool}$: es gibt

- $\binom{n}{2}$ viele Zustandspaarungen (**Beachte:** $\equiv$ ist reflexiv und symmetrisch!) und
- 2 unterschiedliche Symbole in Σ_{Bool}

Damit erhalten wir

$$\begin{aligned} T(n) &= \overbrace{\binom{n}{2}}^{\#\text{ Paare}} \cdot \underbrace{2}_{\#\text{ Symbole}} \\ &= n^2 - n \end{aligned}$$

Diese Funktion wächst quadratisch mit der Zustandszahl - besser als 1. Versuch (exponentielles Wachstum)!

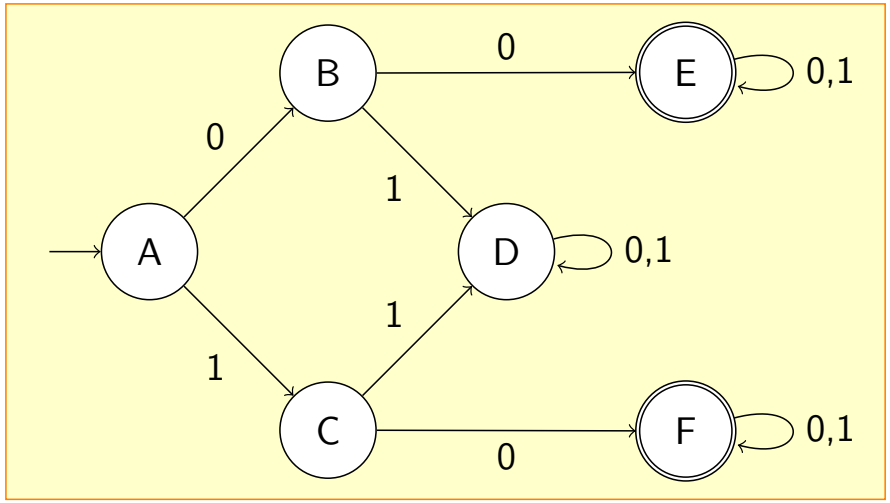
3. Versuch - jetzt wird es fix! (1)

Idee:

- teile Zustände in **mögliche** Äquivalenzklassen ein
  Eine Klasse mit den akzeptierenden, eine mit den verwerfenden Zuständen
- Versuche die Einteilung zu belegen; teile Klassen ggf. auf
  Zustände einer Klasse müssen sich "gleich"verhalten ...

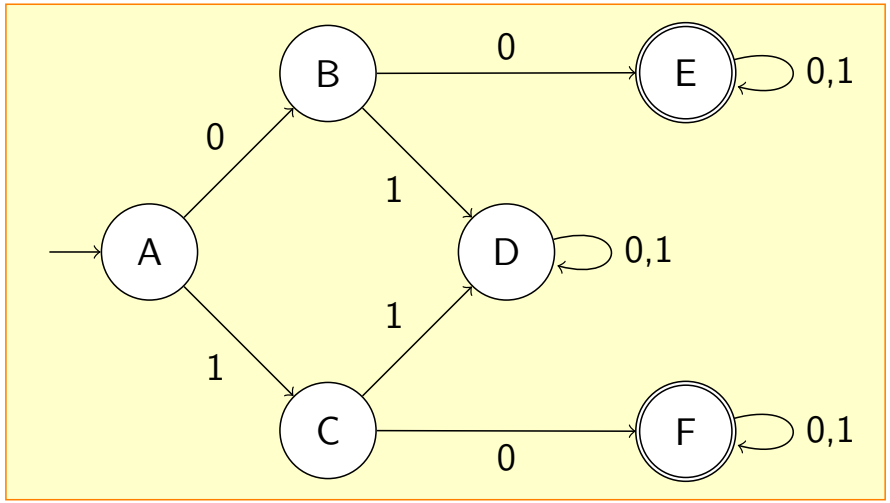
3. Versuch - jetzt wird es fix! (2)

Eq	Q	0	1
[1]	A	[1]	[1]
	B	[2]	[1]
	C	[2]	[1]
	D	[1]	[1]
[2]	E	[2]	[2]
	F	[2]	[2]



3. Versuch - jetzt wird es fix! (3)

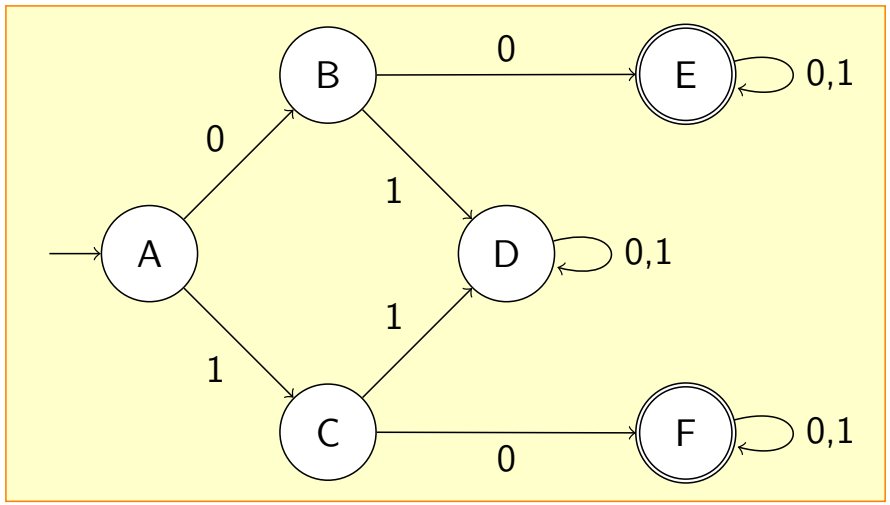
Eq	Q	0	1
[1]	A	[1]	[1]
	B	[2]	[1]
	C	[2]	[1]
	D	[1]	[1]
[2]	E	[2]	[2]
	F	[2]	[2]



- **[2]**: Gleiches Verhalten von E und F
  ⇒ **[2] beibehalten**!
- **[1]**: A, D und B, C verhalten sich unterschiedlich
  ⇒ **[1] aufspalten**!

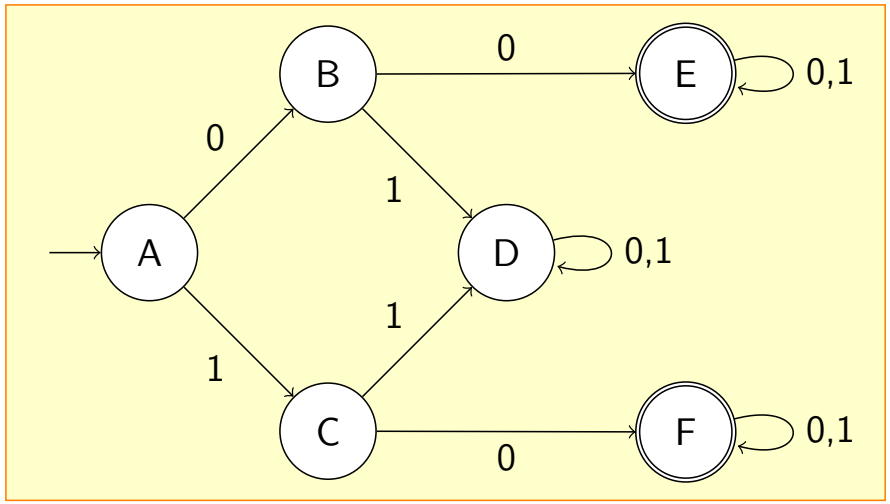
3. Versuch - jetzt wird es fix! (4)

Eq	Q	0	1
[1]	A D	[2] [1]	[2] [1]
[2]	B C	[3] [3]	[1] [1]
[3]	E F	[3] [3]	[3] [3]



3. Versuch - jetzt wird es fix! (5)

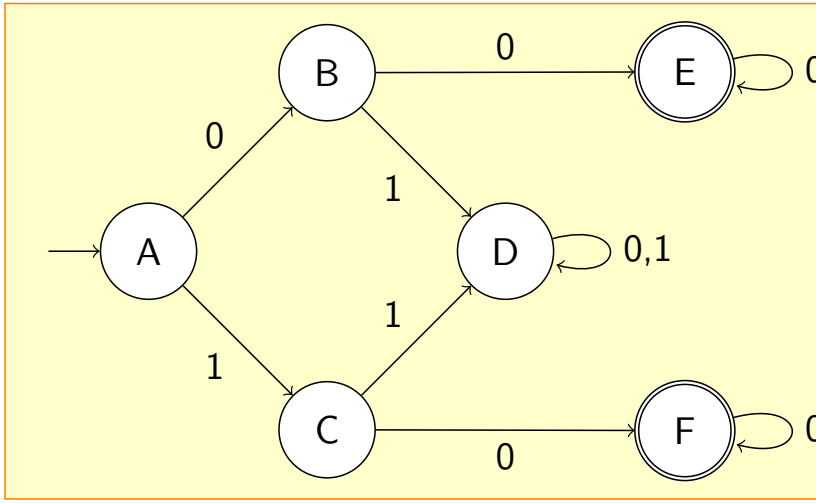
Eq	Q	0	1
[1]	A	[2]	[2]
	D	[1]	[1]
[2]	B	[3]	[1]
	C	[3]	[1]
[3]	E	[3]	[3]
	F	[3]	[3]



- **[3]**: Gleiches Verhalten von E und F
  ⇒ **[3] beibehalten**!
- **[2]**: Gleiches Verhalten von B und C
  ⇒ **[2] beibehalten**!
- **[1]**: A und D verhalten sich unterschiedlich
  ⇒ **[1] aufspalten**!

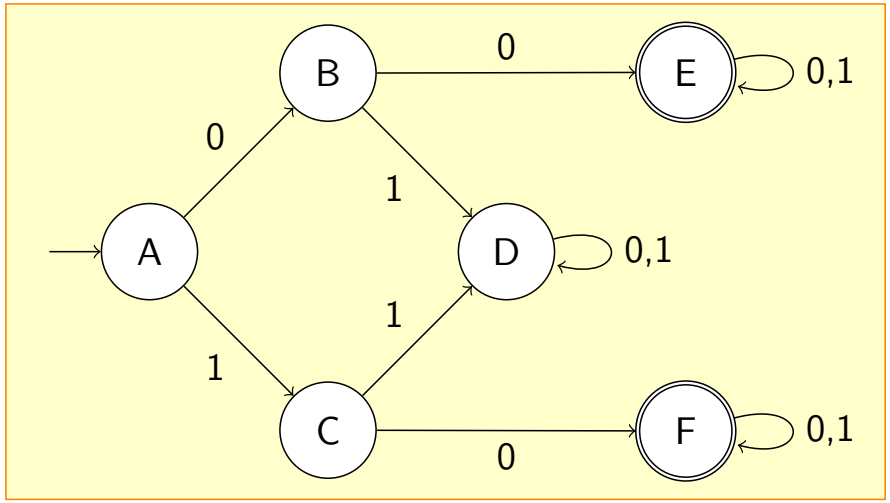
3. Versuch - jetzt wird es fix! (6)

Eq	Q	0	1
[1]	A	[3]	[3]
[2]	D	[2]	[2]
[3]	B	[4]	[2]
	C	[4]	[2]
[4]	E	[4]	[4]
	F	[4]	[4]



3. Versuch - jetzt wird es fix! (7)

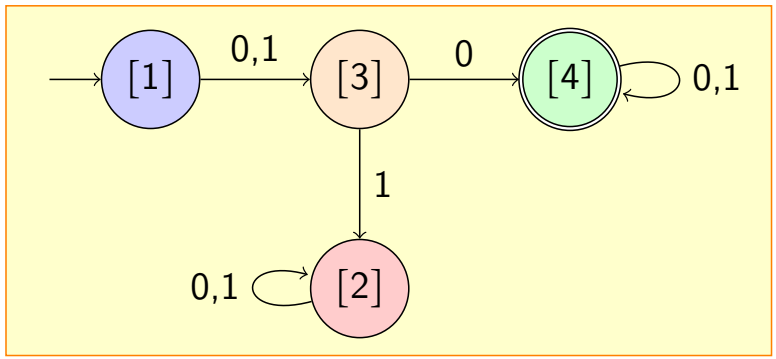
Eq	Q	0	1
[1]	A	[3]	[3]
[2]	D	[2]	[2]
[3]	B C	[4] [4]	[2] [2]
[4]	E F	[4] [4]	[4] [4]



⇒ **Kein Aufspalten mehr erforderlich!**

Minimaler Automat:

- Äquivalenzklassen als neue Zustände
- Obige Tabelle als »Transitionstabelle« für δ
- Startzustand wird Klasse mit altem Startzustand A
- Klassen, die nur akzeptierende Zustände enthalten, werden akzeptierender Zustand



3. Versuch: Beurteilung der Effizienz und Zusammenfassung

Wie viele Tests müssen wir durchführen?

Sei $n = |Q|$; betrachten wir den Fall $\Sigma = \Sigma_{Bool}$:

- $2 \cdot n$ Tests pro Tabelle
- n Tabellen (im *worst case*)
  Aber: Optimierung möglich (Algorithmus von Hopcroft) dann $n \cdot \log_2(n)$!

Abschätzung der Anzahl der Tests $\Sigma = \Sigma_{Bool}$ und $n = \|Q\|$		
1. Versuch *nicht vollendet*	**2. Versuch** *Abhängigkeitsgraph*	**3. Versuch + Optimierung** *Klassen sukzessive spalten*
2^n	n^2	$n \cdot \log_2(n)$