university of groningen

Languages and Machines

L6: FSM Minimization

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

May 4, 2023

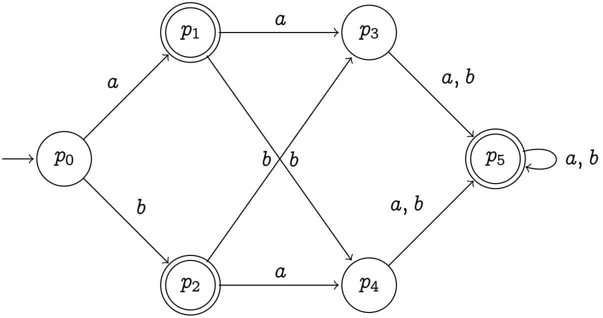
Languages and Their Machines

Regular	$\leftrightarrow$	**Finite State Machines (FSMs)**
Context-free	$\leftrightarrow$	Pushdown Machines
Context-sensitive	$\leftrightarrow$	Linearly-bounded Machines
Decidable	$\leftrightarrow$	Always-terminating Turing Machines
Semi-decidable	$\leftrightarrow$	Turing Machines

Outline

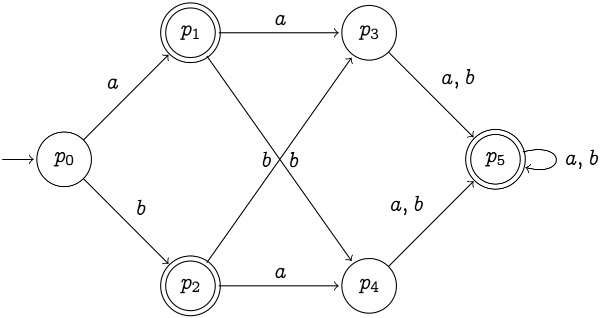
FSM Minimization

Given an FSM M :



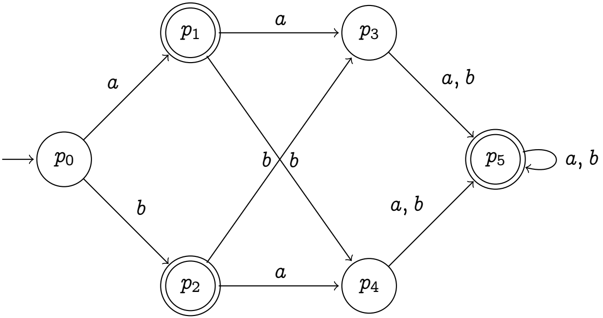
FSM Minimization

Given an FSM M for the language
$L = \{a, b\} \cup \{w \mid w \in \{a, b\}^* \wedge |w| \geq 3\}$:

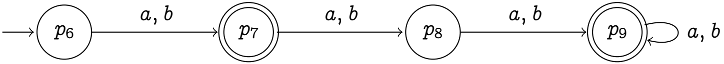


FSM Minimization

Given an FSM M for the language
$L = \{a, b\} \cup \{w \mid w \in \{a, b\}^* \wedge |w| \geq 3\}$:

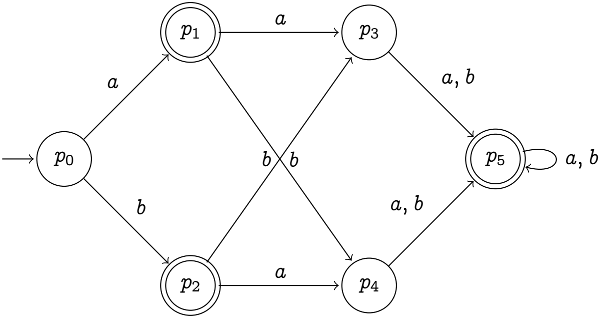


We will minimize M into the *equivalent* FSM M':

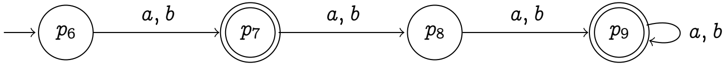


FSM Minimization

Given an FSM M for the language
$L = \{a, b\} \cup \{w \mid w \in \{a, b\}^* \wedge |w| \geq 3\}$:



We will minimize M into the *equivalent* FSM M':



Here 'equivalence' means $L(M) = L(M') = L$.

Minimization

We want to minimize an (D)FSM M into M'

- Minimizing M means **collapsing** some of its states
- M' is a different (but smaller) machine that accepts $L(M)$
- Key idea: Never collapse an accept state and a reject state

Minimization

We want to minimize an (D)FSM M into M'

- Minimizing M means **collapsing** some of its states
- M' is a different (but smaller) machine that accepts $L(M)$
- Key idea: Never collapse an accept state and a reject state

Observations:

1. If $p = \hat{\delta}(q_0, x) \in F$ and $q = \hat{\delta}(q_0, y) \notin F$.
   String x must be accepted but y must be rejected, even after collapsing.

Minimization

We want to minimize an (D)FSM M into M'

- Minimizing M means **collapsing** some of its states
- M' is a different (but smaller) machine that accepts $L(M)$
- Key idea: Never collapse an accept state and a reject state

Observations:

1. If $p = \hat{\delta}(q_0, x) \in F$ and $q = \hat{\delta}(q_0, y) \notin F$.
   String x must be accepted but y must be rejected, even after collapsing.
2. If p and q are collapsed then, to maintain determinism, also $\delta(p, a)$ and $\delta(q, a)$ must be collapsed.

Minimization

We want to minimize an (D)FSM M into M'

- Minimizing M means **collapsing** some of its states
- M' is a different (but smaller) machine that accepts $L(M)$
- Key idea: Never collapse an accept state and a reject state

Observations:

1. If $p = \hat{\delta}(q_0, x) \in F$ and $q = \hat{\delta}(q_0, y) \notin F$.
   String x must be accepted but y must be rejected, even after collapsing.
2. If p and q are collapsed then, to maintain determinism, also $\delta(p, a)$ and $\delta(q, a)$ must be collapsed.

$\Rightarrow$ If $\hat{\delta}(p, x) \in F$ and $\hat{\delta}(q, x) \notin F$, for some $x \in \Sigma^*$, then we cannot collapse p and q.
Otherwise, p and q can be collapsed.

Minimization, Formally

- Minimization depends on an equivalence relation $\approx$, defined on the set of states Q of the given FSM M.
- Intuitively, $\approx$ formalizes which states should (not) be collapsed.

Minimization, Formally

- Minimization depends on an equivalence relation $\approx$, defined on the set of states Q of the given FSM M.
- Intuitively, $\approx$ formalizes which states should (not) be collapsed.
- Using $\approx$, we will define a minimized variant of M, denoted $M/_{\approx}$ (the 'quotient' of M).

Minimization, Formally

- Minimization depends on an equivalence relation $\approx$, defined on the set of states Q of the given FSM M.
- Intuitively, $\approx$ formalizes which states should (not) be collapsed.
- Using $\approx$, we will define a minimized variant of M, denoted $M/_{\approx}$ (the 'quotient' of M).
- We give a procedure to compute $\approx$.

Minimization, Formally

- Minimization depends on an equivalence relation $\approx$, defined on the set of states Q of the given FSM M.
- Intuitively, $\approx$ formalizes which states should (not) be collapsed.
- Using $\approx$, we will define a minimized variant of M, denoted $M/_{\approx}$ (the 'quotient' of M).
- We give a procedure to compute $\approx$.

We first recall key notions about equivalences and their partitions.

Outline

Equivalence Relations

Let $\mathcal{R}$ be a relation on a set S.

- $\mathcal{R}$ is an equivalence if it is reflexive, symmetric, and transitive.

Equivalence Relations

Let $\mathcal{R}$ be a relation on a set S.

- $\mathcal{R}$ is an equivalence if it is reflexive, symmetric, and transitive.
- That is, an equivalence $\mathcal{R}$ on S satisfies:
  - $s \mathrel{\mathcal{R}} s$ for all $s \in S$.
  - $s \mathrel{\mathcal{R}} t$ implies $t \mathrel{\mathcal{R}} s$.
  - $s \mathrel{\mathcal{R}} t$ and $t \mathrel{\mathcal{R}} u$ imply $s \mathrel{\mathcal{R}} u$.

Equivalence Relations

Let $\mathcal{R}$ be a relation on a set S.

- $\mathcal{R}$ is an equivalence if it is reflexive, symmetric, and transitive.
- That is, an equivalence $\mathcal{R}$ on S satisfies:
  - $s \,\mathcal{R}\, s$ for all $s \in S$.
  - $s \,\mathcal{R}\, t$ implies $t \,\mathcal{R}\, s$.
  - $s \,\mathcal{R}\, t$ and $t \,\mathcal{R}\, u$ imply $s \,\mathcal{R}\, u$.

Examples:

- The '$=$' relation is an equivalence on $\mathbb{R}$.
- The '$<$' relation is *not* an equivalence on $\mathbb{R}$ (why?).
- Define the relation $\equiv_2$ on $\mathbb{Z}$:

$$a \equiv_2 b \iff a \text{ and } b \text{ have the same remainder when divided by } 2$$

E.g., $42 \equiv_2 24$, $0 \equiv_2 24$, $13 \equiv_2 31$, $1 \equiv_2 13$, but $13 \not\equiv_2 24$.

Equivalence Relations

Let $\mathcal{R}$ be a relation on a set S.

- $\mathcal{R}$ is an equivalence if it is reflexive, symmetric, and transitive.
- That is, an equivalence $\mathcal{R}$ on S satisfies:
  - $s \,\mathcal{R}\, s$ for all $s \in S$.
  - $s \,\mathcal{R}\, t$ implies $t \,\mathcal{R}\, s$.
  - $s \,\mathcal{R}\, t$ and $t \,\mathcal{R}\, u$ imply $s \,\mathcal{R}\, u$.

Examples:

- The '$=$' relation is an equivalence on $\mathbb{R}$.
- The '$<$' relation is *not* an equivalence on $\mathbb{R}$ (why?).
- Define the relation $\equiv_2$ on $\mathbb{Z}$:

  $a \equiv_2 b \iff a$ and b have the same remainder when divided by 2

  E.g., $42 \equiv_2 24$, $0 \equiv_2 24$, $13 \equiv_2 31$, $1 \equiv_2 13$, but $13 \not\equiv_2 24$.
  Is relation $\equiv_2$ is an equivalence?

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set

$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}t\}$$

- This way, e.g., the equivalence classes of '$\equiv_2$':

$$[0] \quad = \{a \in \mathbb{Z} \mid a \equiv_2 0\}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':

$$[0] \quad = \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned} [0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\ [1] &= \end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s \, \mathcal{R} \, t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':
$$\begin{aligned} [0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\ [1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} \end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned}
[0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\
[1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} = \{1, 3, 5, \ldots\} \\
[2] &=
\end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned}
[0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\
[1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} = \{1, 3, 5, \ldots\} \\
[2] &= \{a \in \mathbb{Z} \mid a \equiv_2 2\}
\end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned}
[0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\
[1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} = \{1, 3, 5, \ldots\} \\
[2] &= \{a \in \mathbb{Z} \mid a \equiv_2 2\} = \{0, 2, 4, \ldots\}
\end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set

$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$

- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned}
[0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\
[1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} = \{1, 3, 5, \ldots\} \\
[2] &= \{a \in \mathbb{Z} \mid a \equiv_2 2\} = \{0, 2, 4, \ldots\} \\
[3] &=
\end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set

$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$

- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned}
[0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\
[1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} = \{1, 3, 5, \ldots\} \\
[2] &= \{a \in \mathbb{Z} \mid a \equiv_2 2\} = \{0, 2, 4, \ldots\} \\
[3] &=
\end{aligned}$$

Equivalence Classes

- Given an equivalence $\mathcal{R}$ on S, the equivalence class of $s \in S$ is the set
$$[s] = \mathcal{R}(s) = \{t \in S \mid s\,\mathcal{R}\,t\}$$
- This way, e.g., the equivalence classes of '$\equiv_2$':

$$\begin{aligned}
[0] &= \{a \in \mathbb{Z} \mid a \equiv_2 0\} = \{0, 2, 4, \ldots\} \\
[1] &= \{b \in \mathbb{Z} \mid b \equiv_2 1\} = \{1, 3, 5, \ldots\} \\
[2] &= \{a \in \mathbb{Z} \mid a \equiv_2 2\} = \{0, 2, 4, \ldots\} \\
[3] &= \{b \in \mathbb{Z} \mid b \equiv_2 3\} = \{1, 3, 5, \ldots\} \\
&\vdots
\end{aligned}$$

Notice: $[0] = [2] = [4] \ldots$ and $[1] = [3] = [5] \ldots$.
'$\equiv_2$' defines two equivalence classes, for even and odd integers.

Equivalence Classes and Partitions

- Given some $S \neq \emptyset$, a partition $\mathcal{P}$ is a set $\{P_1, P_2, \ldots, P_n\}$ consisting of non-empty subsets of S, called **blocks**,

Equivalence Classes and Partitions

- Given some $S \neq \emptyset$, a partition $\mathcal{P}$ is a set $\{P_1, P_2, \ldots, P_n\}$ consisting of non-empty subsets of S, called **blocks**,where:
  1. The partition $\mathcal{P}$ *covers* S. That is: $P_1 \cup P_2 \cup \cdots \cup P_n = S$.

Equivalence Classes and Partitions

- Given some $S \neq \emptyset$, a partition $\mathcal{P}$ is a set $\{P_1, P_2, \ldots, P_n\}$ consisting of non-empty subsets of S, called **blocks**,where:
  1. The partition $\mathcal{P}$ *covers* S. That is: $P_1 \cup P_2 \cup \cdots \cup P_n = S$.
  2. The blocks are *disjoint*: $P_i \cap P_j = \emptyset$, for all $P_i, P_j \in \mathcal{P}$ with $i \neq j$.

Two useful theorems.

1. **Equivalences induce partitions**
   Let $\mathcal{R}$ be an equivalence on S. Let $\mathcal{P}$ be the set of all equivalence classes $[s]$, with $s \in S$. Then $\mathcal{P}$ is a partition of S.

Equivalence Classes and Partitions

- Given some $S \neq \emptyset$, a partition $\mathcal{P}$ is a set $\{P_1, P_2, \ldots, P_n\}$ consisting of non-empty subsets of S, called **blocks**,where:
  1. The partition $\mathcal{P}$ *covers* S. That is: $P_1 \cup P_2 \cup \cdots \cup P_n = S$.
  2. The blocks are *disjoint*: $P_i \cap P_j = \emptyset$, for all $P_i, P_j \in \mathcal{P}$ with $i \neq j$.

Two useful theorems.

1. **Equivalences induce partitions**
   Let $\mathcal{R}$ be an equivalence on S. Let $\mathcal{P}$ be the set of all equivalence classes $[s]$, with $s \in S$. Then $\mathcal{P}$ is a partition of S.
2. **Partitions induce equivalences**
   Let $\mathcal{P} = \{P_i \mid i \in I\}$ be a partition of S. Define the relation $\mathcal{R}_\mathcal{P}$:

   $$a \, \mathcal{R}_\mathcal{P} \, b \iff a \text{ and } b \text{ belong to the same block of } \mathcal{P}$$

   $\mathcal{R}_\mathcal{P}$ is an equivalence; the blocks P_i are its equivalence classes.

Outline

Minimization, Formally

- Define the relation $\approx$ on Q, which will determine collapsing:

$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

Because of '$\forall$', even a single x serves to prove non-equivalence.

Minimization, Formally

- Define the relation $\approx$ on Q, which will determine collapsing:

$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

Because of '$\forall$', even a single x serves to prove non-equivalence.

Minimization, Formally

- Define the relation $\approx$ on Q, which will determine collapsing:

$$\boxed{p \approx q \iff \forall x \in \Sigma^* \left(\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F\right)}$$

  Because of '$\forall$', even a single x serves to prove non-equivalence.
- Relation $\approx$ is an equivalence. Convince yourself!

Minimization, Formally

- Define the relation $\approx$ on Q, which will determine collapsing:

$$\boxed{p \approx q \iff \forall x \in \Sigma^* \,(\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

  Because of '$\forall$', even a single x serves to prove non-equivalence.
- Relation $\approx$ is an equivalence. Convince yourself!
- Equivalence class for state p:

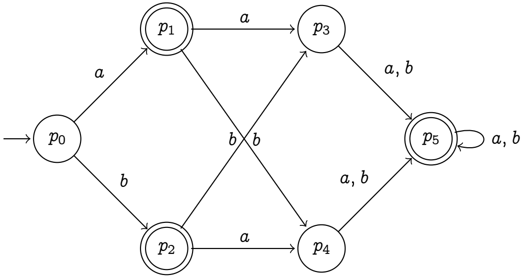
$$[p] = \{q \mid q \approx p\}$$

- Every $p \in Q$ is contained in one class $[p]$ and

$$p \approx q \iff [p] = [q]$$

- Every equivalence class is a subset of F or disjoint from F.

The Relation $\approx$ on States

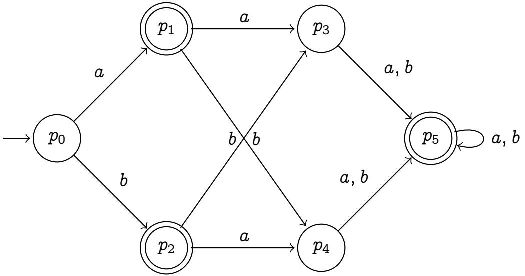


$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:

$[p_5] =$

The Relation $\approx$ on States

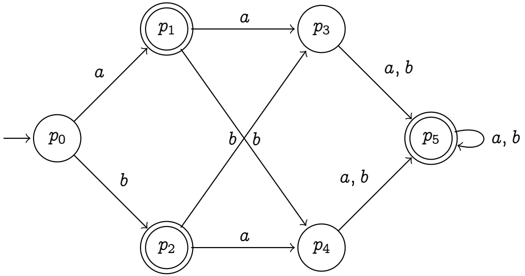


$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:

$[p_5] = \{p_5\}$,

The Relation $\approx$ on States

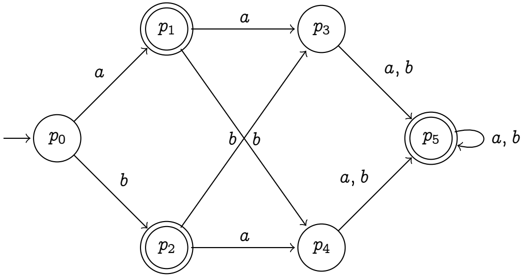


$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:

$[p_5] = \{p_5\}, \; [p_3] =$

The Relation $\approx$ on States

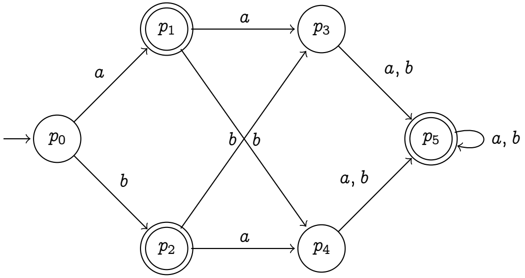


$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:

$[p_5] = \{p_5\}, \; [p_3] = \{p_3, p_4\}$

The Relation $\approx$ on States

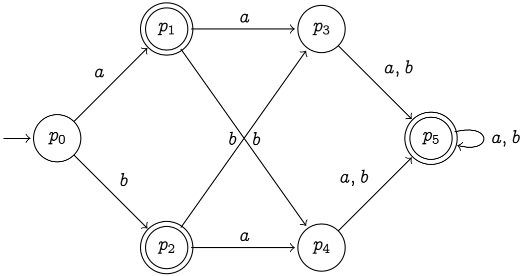


$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:

$[p_5] = \{p_5\}$, $[p_3] = \{p_3, p_4\} = [p_4]$,

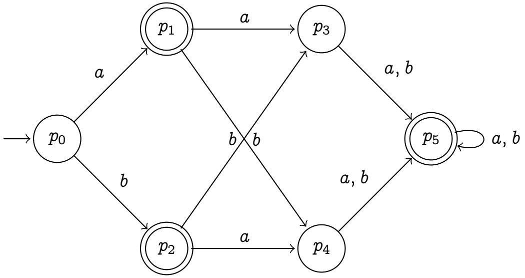
The Relation $\approx$ on States



$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:
$[p_5] = \{p_5\}$, $[p_3] = \{p_3, p_4\} = [p_4]$, $[p_1] = [p_2] = \{p_1, p_2\}$, $[p_0] = \{p_0\}$.

The Relation $\approx$ on States



$$\boxed{p \approx q \iff \forall x \in \Sigma^* \, (\hat{\delta}(p, x) \in F \iff \hat{\delta}(q, x) \in F)}$$

We have:
$[p_5] = \{p_5\}$, $[p_3] = \{p_3, p_4\} = [p_4]$, $[p_1] = [p_2] = \{p_1, p_2\}$, $[p_0] = \{p_0\}$.

We also have, e.g., $p_2 \not\approx p_3$, because $\hat{\delta}(p_3, a) \in F$ but $\hat{\delta}(p_2, a) \notin F$.

Outline

Quotient FSM

Given $M = (Q, \Sigma, \delta, q_0, F)$, we define $M/_{\approx}$: the quotient FSM of M:

$$M/_{\approx} = (Q', \Sigma, \delta', q_0', F')$$

where:

$$\begin{aligned} Q' &= \{[p] \mid p \in Q\} \\ \delta'([p], a) &= [\delta(p, a)] \\ q_0' &= [q_0] \\ F' &= \{[p] \mid p \in F\} \end{aligned}$$

Quotient FSM

Given $M = (Q, \Sigma, \delta, q_0, F)$, we define $M/_{\approx}$: the quotient FSM of M:

$$M/_{\approx} = (Q', \Sigma, \delta', q_0', F')$$

where:

$$\begin{aligned} Q' &= \{[p] \mid p \in Q\} \\ \delta'([p], a) &= [\delta(p, a)] \\ q_0' &= [q_0] \\ F' &= \{[p] \mid p \in F\} \end{aligned}$$

- The states of $M/_{\approx}$ are the equivalence classes of $\approx$
- We must show: (i) $M/_{\approx}$ is well-defined and (ii) $L(M) = L(M/_{\approx})$.

Quotient FSM

Given $M = (Q, \Sigma, \delta, q_0, F)$, we define $M/_{\approx}$: the quotient FSM of M:

$$M/_{\approx} = (Q', \Sigma, \delta', q'_0, F')$$

where:

$$\begin{aligned} Q' &= \{[p] \mid p \in Q\} \\ \delta'([p], a) &= [\delta(p, a)] \\ q'_0 &= [q_0] \\ F' &= \{[p] \mid p \in F\} \end{aligned}$$

- The states of $M/_{\approx}$ are the equivalence classes of $\approx$
- We must show: (i) $M/_{\approx}$ is well-defined and (ii) $L(M) = L(M/_{\approx})$.
- Some crucial properties for (i):
  (P1) $p \approx q$ implies $\delta(p, a) \approx \delta(q, a)$.
  That is: $[p] = [q]$ implies $[\delta(p, a)] = [\delta(q, a)]$
  (P2) $p \in F \iff [p] \in F'$.

Properties of $M/_\approx$

We have $\forall x \in \Sigma^*$, $\hat{\delta}'([p], x) = [\hat{\delta}(p, x)]$.
The proof is by induction on x:

- Base case, $x = \epsilon$:

$$\begin{aligned} \hat{\delta}'([p], \epsilon) &= [p] && \text{Def of } \delta' \\ &= [\hat{\delta}(p, \epsilon)] && \text{Def of } \delta \end{aligned}$$

- Inductive step (with $a \in \Sigma$):

$$\begin{aligned} \hat{\delta}'([p], xa) &= \delta'(\hat{\delta}'([p], x), a) && \text{Def of } \delta' \\ &= \delta'([\hat{\delta}(p, x)], a) && \text{IH} \\ &= [\delta(\hat{\delta}(p, x), a)] && \text{Def of } \delta' \\ &= [\hat{\delta}(p, xa)] && \text{Def of } \hat{\delta} \end{aligned}$$

Properties of $M/_{\approx}$

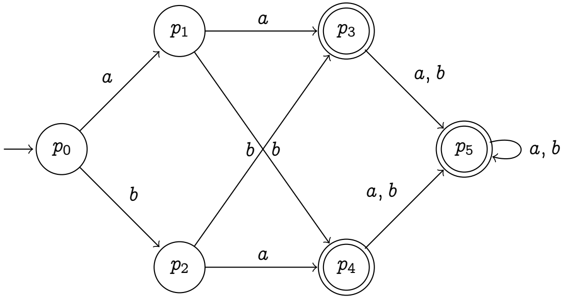
We have $L(M) = L(M/_{\approx})$

For the proof, suppose $x \in \Sigma^*$.

$$
\begin{aligned}
x \in L(M/_{\approx}) &\iff \hat{\delta}'(q_0', x) \in F' && \text{Def of acceptance} \\
&\iff \hat{\delta}'([q_0], x) \in F' && \text{Def of } q_0' \\
&\iff [\hat{\delta}(q_0, x)] \in F' && \text{Previous slide} \\
&\iff \hat{\delta}(q_0, x) \in F && \text{Well-defined acceptance (P2)} \\
&\iff x \in L(M) && \text{Acceptance}
\end{aligned}
$$

A Slightly Modified FSM

Consider the FSM:

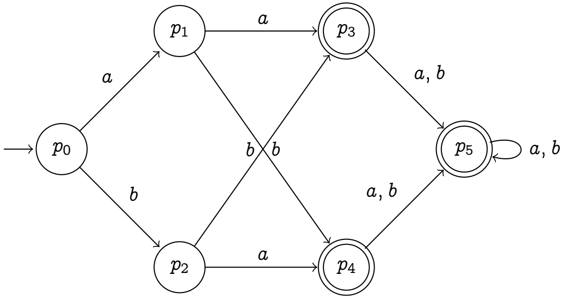


Same transitions as before, here with different accepting states.

- What is $L(M)$?

A Slightly Modified FSM

Consider the FSM:

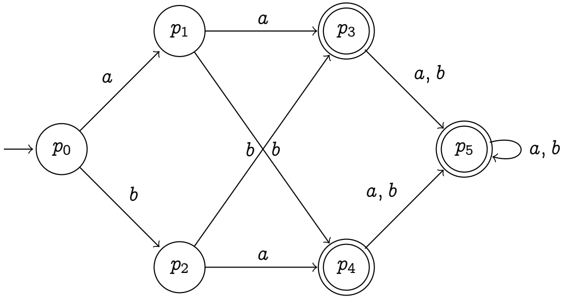


Same transitions as before, here with different accepting states.

- What is $L(M)$?
  The language $L = \{w \mid |w| \geq 2\}$

A Slightly Modified FSM

Consider the FSM:

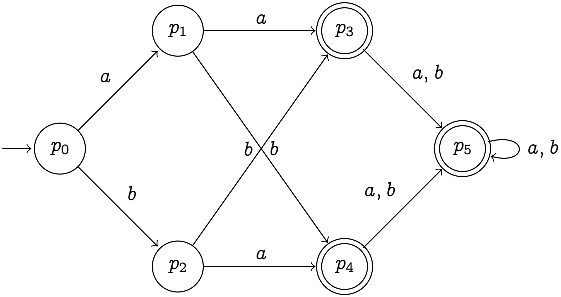


Same transitions as before, here with different accepting states.

- What is $L(M)$?
  The language $L = \{w \mid |w| \geq 2\}$
- What is $M/_{\approx}$?

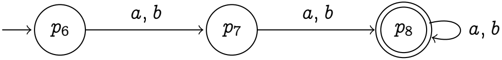
A Slightly Modified FSM

Consider the FSM:



Same transitions as before, here with different accepting states.

- What is $L(M)$?
  The language $L = \{w \mid |w| \geq 2\}$
- What is $M/_{\approx}$?



Outline

How to compute $\approx$?

We have defined $M/_{\approx}$ but how to compute $\approx$?

How to compute $\approx$?

We have defined $M/_{\approx}$ but how to compute $\approx$?

Consider a DFSM with no inaccessible states.
The following algorithm computes $\approx$ by analyzing pairs of states.
We mark $\{p, q\}$ when we discover that p and q are not equivalent.

How to compute $\approx$?

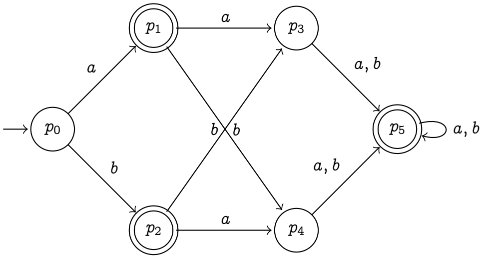
We have defined $M/_\approx$ but how to compute $\approx$?

Consider a DFSM with no inaccessible states.
The following algorithm computes $\approx$ by analyzing pairs of states.
We mark $\{p, q\}$ when we discover that p and q are not equivalent.

The algorithm:

1. Write down a table with all pairs $\{p, q\}$, initially unmarked.
2. Mark $\{p, q\}$ if $p \in F$ and $q \notin F$ (or viceversa).
3. Repeat the following, until no more changes occur:
   If there is an unmarked pair $\{p, q\}$ such that the pair $\{\delta(p, a), \delta(q, a)\}$ is marked (for some $a \in \Sigma$) then mark $\{p, q\}$.
4. At the end, we have that $p \approx q$ iff $\{p, q\}$ is not marked.

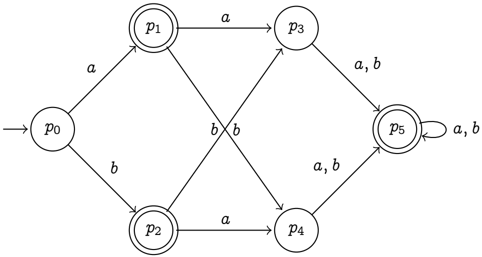
Example (1/2)



Step 1 (initialization):

0					
-	1∗				
-	-	2∗			
-	-	-	3		
-	-	-	-	4	
-	-	-	-	-	5∗

Example (1/2)



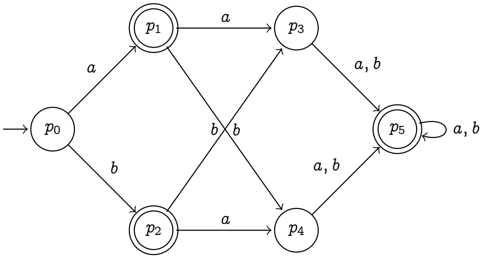
Step 1 (initialization):

0					
-	1$*$				
-	-	2$*$			
-	-	-	3		
-	-	-	-	4	
-	-	-	-	-	5$*$

Step 2 (initial marking):

0					
$\checkmark$	1$*$				
$\checkmark$	-	2$*$			
-	$\checkmark$	$\checkmark$	3		
-	$\checkmark$	$\checkmark$	-	4	
$\checkmark$	-	-	$\checkmark$	$\checkmark$	5$*$

Example (1/2)



Step 1 (initialization):

0					
-	1*				
-	-	2*			
-	-	-	3		
-	-	-	-	4	
-	-	-	-	-	5*

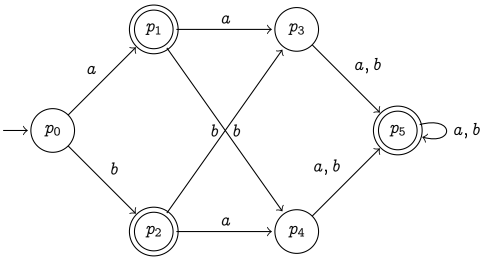
Step 2 (initial marking):

0					
✓	1*				
✓	-	2*			
-	✓	✓	3		
-	✓	✓	-	4	
✓	-	-	✓	✓	5*

Step 3 (first pass):

0					
✓	1*				
✓	-	2*			
-	✓	✓	3		
-	✓	✓	-	4	
✓	✓	✓	✓	✓	5*

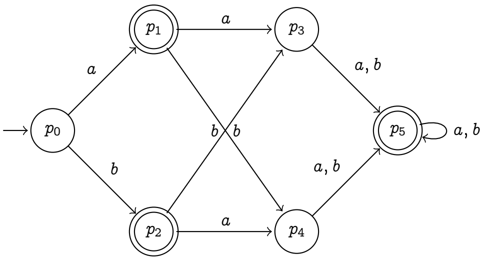
Example (2/2)



Step 3 (first pass, prev slide):

0					
✓	1∗				
✓	-	2∗			
-	✓	✓	3		
-	✓	✓	-	4	
✓	✓	✓	✓	✓	5∗

Example (2/2)



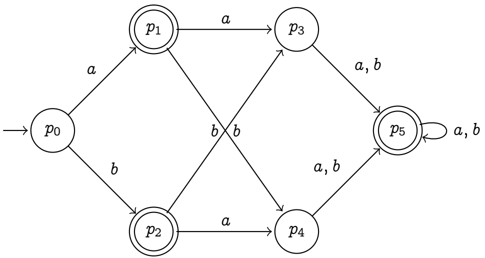
Step 3 (first pass, prev slide):

0					
✓	1∗				
✓	-	2∗			
-	✓	✓	3		
-	✓	✓	-	4	
✓	✓	✓	✓	✓	5∗

Step 3 (second pass):

0					
✓	1∗				
✓	-	2∗			
✓	✓	✓	3		
✓	✓	✓	-	4	
✓	✓	✓	✓	✓	5∗

Example (2/2)



Step 3 (first pass, prev slide):

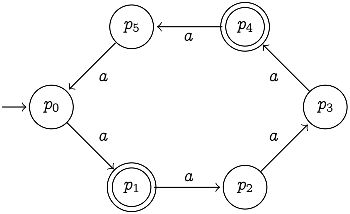
0					
✓	1∗				
✓	-	2∗			
-	✓	✓	3		
-	✓	✓	-	4	
✓	✓	✓	✓	✓	5∗

Step 3 (second pass):

0					
✓	1∗				
✓	-	2∗			
✓	✓	✓	3		
✓	✓	✓	-	4	
✓	✓	✓	✓	✓	5∗

At this point, the remaining unmarked pairs are $\{1, 2\}$ and $\{3, 4\}$. They lead to unmarked pairs, indicating that $p_1 \approx p_2$ and $p_3 \approx p_4$.

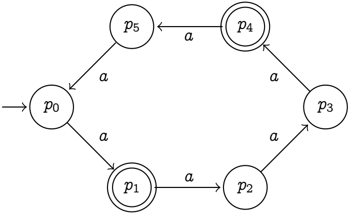
Another Example (1/2)



Step 2 (initial marking):

0					
✓	1∗				
-	✓	2			
-	✓	-	3		
✓	-	✓	✓	4∗	
-	✓	-	-	✓	5

Another Example (1/2)



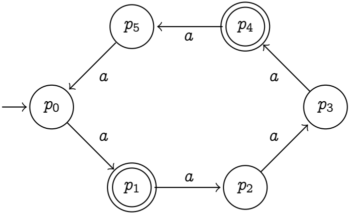
Step 3 (analysis for the first pass):

- $\{0, 2\} \to \{1, 3\}$, so $\{0, 2\}$ is marked

Step 2 (initial marking):

0					
✓	1$*$				
-	✓	2			
-	✓	-	3		
✓	-	✓	✓	4$*$	
-	✓	-	-	✓	5

Another Example (1/2)



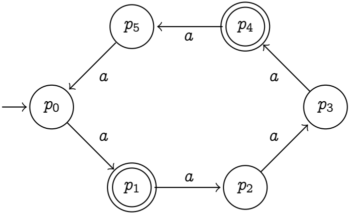
Step 3 (analysis for the first pass):

- $\{0, 2\} \to \{1, 3\}$, so $\{0, 2\}$ is marked
- $\{0, 3\} \to \{1, 4\}$, so $\{0, 3\}$ stays unmarked

Step 2 (initial marking):

0					
✓	1∗				
-	✓	2			
-	✓	-	3		
✓	-	✓	✓	4∗	
-	✓	-	-	✓	5

Another Example (1/2)



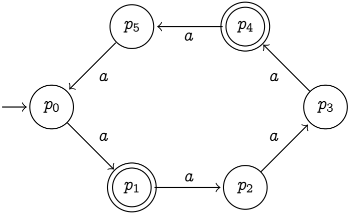
Step 2 (initial marking):

0					
✓	1∗				
-	✓	2			
-	✓	-	3		
✓	-	✓	✓	4∗	
-	✓	-	-	✓	5

Step 3 (analysis for the first pass):

- $\{0,2\} \to \{1,3\}$, so $\{0,2\}$ is marked
- $\{0,3\} \to \{1,4\}$, so $\{0,3\}$ stays unmarked
- $\{0,5\} \to \{1,0\}$, so $\{0,5\}$ is marked

Another Example (1/2)



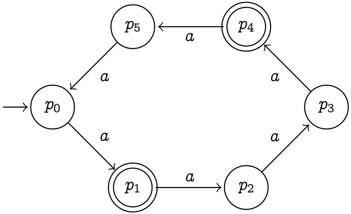
Step 2 (initial marking):

0					
✓	1∗				
-	✓	2			
-	✓	-	3		
✓	-	✓	✓	4∗	
-	✓	-	-	✓	5

Step 3 (analysis for the first pass):

- $\{0,2\} \to \{1,3\}$, so $\{0,2\}$ is marked
- $\{0,3\} \to \{1,4\}$, so $\{0,3\}$ stays unmarked
- $\{0,5\} \to \{1,0\}$, so $\{0,5\}$ is marked
- $\{1,4\} \to \{2,5\}$, so $\{1,4\}$ stays unmarked
- $\{2,3\} \to \{3,4\}$, so $\{2,3\}$ is marked
- $\{2,5\} \to \{0,3\}$, so $\{2,5\}$ stays unmarked
- $\{3,5\} \to \{0,4\}$, so $\{3,5\}$ is marked

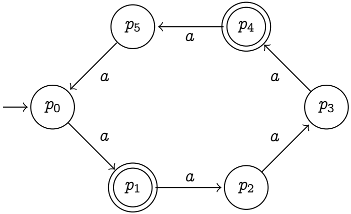
Another Example (2/2)



Step 3 (analysis for the first pass, prev slide):

- $\{0, 2\} \rightarrow \{1, 3\}$, so $\{0, 2\}$ is marked
- $\{0, 3\} \rightarrow \{1, 4\}$, so $\{0, 3\}$ stays unmarked
- $\{0, 5\} \rightarrow \{1, 0\}$, so $\{0, 5\}$ is marked
- $\{1, 4\} \rightarrow \{2, 5\}$, so $\{1, 4\}$ stays unmarked
- $\{2, 3\} \rightarrow \{3, 4\}$, so $\{2, 3\}$ is marked
- $\{2, 5\} \rightarrow \{0, 3\}$, so $\{2, 5\}$ stays unmarked
- $\{3, 5\} \rightarrow \{0, 4\}$, so $\{3, 5\}$ is marked

Another Example (2/2)



Step 3 (analysis for the first pass, prev slide):

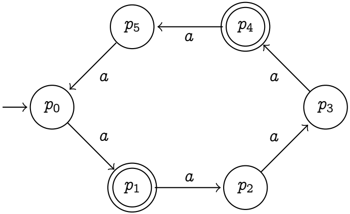
- $\{0,2\} \to \{1,3\}$, so $\{0,2\}$ is marked
- $\{0,3\} \to \{1,4\}$, so $\{0,3\}$ stays unmarked
- $\{0,5\} \to \{1,0\}$, so $\{0,5\}$ is marked
- $\{1,4\} \to \{2,5\}$, so $\{1,4\}$ stays unmarked
- $\{2,3\} \to \{3,4\}$, so $\{2,3\}$ is marked
- $\{2,5\} \to \{0,3\}$, so $\{2,5\}$ stays unmarked
- $\{3,5\} \to \{0,4\}$, so $\{3,5\}$ is marked

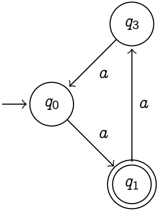
Step 3 (result of the first pass):

0					
✓	1∗				
✓	✓	2			
-	✓	✓	3		
✓	-	✓	✓	4∗	
✓	✓	-	✓	✓	5

A second pass gives $\{0,3\} \to \{1,4\} \to \{2,5\} \to \{0,3\}$. All unmarked: we are done.

Another Example (2/2)





Step 3 (analysis for the first pass, prev slide):

- $\{0,2\} \to \{1,3\}$, so $\{0,2\}$ is marked
- $\{0,3\} \to \{1,4\}$, so $\{0,3\}$ stays unmarked
- $\{0,5\} \to \{1,0\}$, so $\{0,5\}$ is marked
- $\{1,4\} \to \{2,5\}$, so $\{1,4\}$ stays unmarked
- $\{2,3\} \to \{3,4\}$, so $\{2,3\}$ is marked
- $\{2,5\} \to \{0,3\}$, so $\{2,5\}$ stays unmarked
- $\{3,5\} \to \{0,4\}$, so $\{3,5\}$ is marked

Step 3 (result of the first pass):

0					
✓	1∗				
✓	✓	2			
-	✓	✓	3		
✓	-	✓	✓	4∗	
✓	✓	-	✓	✓	5

A second pass gives $\{0,3\} \to \{1,4\} \to \{2,5\} \to \{0,3\}$. All unmarked: we are done.

Hence: $p_0 \approx p_3$, $p_1 \approx p_4$, and $p_2 \approx p_5$.

Outline

Brzozowski's algorithm

Another, less efficient way of obtaining a minimal FSM, using some operations on machines:

- $rev(M)$ denotes the reverse of M
- $det(M)$ denotes the powerset construction applied to M
- $reach(M)$ discards inaccessible states from M

Brzozowski's algorithm

Another, less efficient way of obtaining a minimal FSM, using some operations on machines:

- $rev(M)$ denotes the reverse of M
- $det(M)$ denotes the powerset construction applied to M
- $reach(M)$ discards inaccessible states from M

The algorithm:

1. Obtain $rev(M)$: an NFSM that recognizes the reverse of $L(M)$.

Brzozowski's algorithm

Another, less efficient way of obtaining a minimal FSM, using some operations on machines:

- $rev(M)$ denotes the reverse of M
- $det(M)$ denotes the powerset construction applied to M
- $reach(M)$ discards inaccessible states from M

The algorithm:

1. Obtain $rev(M)$: an NFSM that recognizes the reverse of $L(M)$.
2. Obtain $det(rev(M))$, the deterministic FSM that still recognizes the reverse of $L(M)$.

Brzozowski's algorithm

Another, less efficient way of obtaining a minimal FSM, using some operations on machines:

- $rev(M)$ denotes the reverse of M
- $det(M)$ denotes the powerset construction applied to M
- $reach(M)$ discards inaccessible states from M

The algorithm:

1. Obtain $rev(M)$: an NFSM that recognizes the reverse of $L(M)$.
2. Obtain $det(rev(M))$, the deterministic FSM that still recognizes the reverse of $L(M)$.
3. Obtaining $N = reach(det(rev(M)))$

Brzozowski's algorithm

Another, less efficient way of obtaining a minimal FSM, using some operations on machines:

- $rev(M)$ denotes the reverse of M
- $det(M)$ denotes the powerset construction applied to M
- $reach(M)$ discards inaccessible states from M

The algorithm:

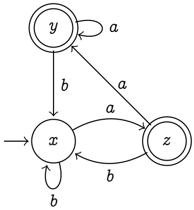
1. Obtain $rev(M)$: an NFSM that recognizes the reverse of $L(M)$.
2. Obtain $det(rev(M))$, the deterministic FSM that still recognizes the reverse of $L(M)$.
3. Obtaining $N = reach(det(rev(M)))$
4. Apply steps 1-3 to N, obtaining an FSM that recognizes $L(M)$:

$$min(M) = reach(det(rev(reach(det(rev(M))))))$$

Brzozowski's algorithm: Example

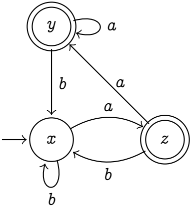
[Adapted from slides by Jan Rutten (2012)]

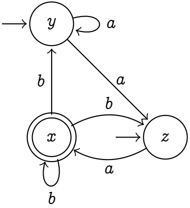
M:



Brzozowski's algorithm: Example

[Adapted from slides by Jan Rutten (2012)]

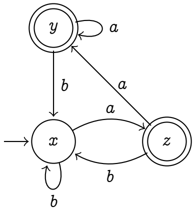
M: 

$rev(M)$: 

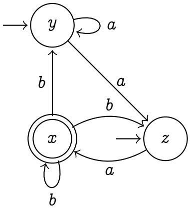
Brzozowski's algorithm: Example

[Adapted from slides by Jan Rutten (2012)]

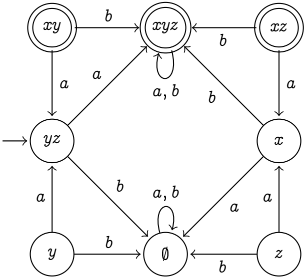
M:



$rev(M)$:



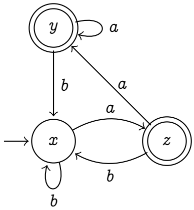
$det(rev(M))$:



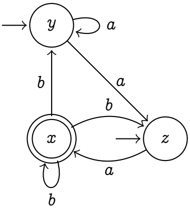
Brzozowski's algorithm: Example

[Adapted from slides by Jan Rutten (2012)]

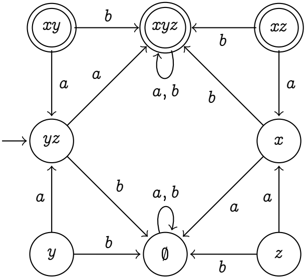
M:



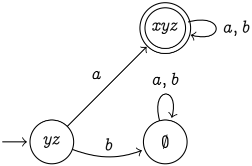
$rev(M)$:



$det(rev(M))$:



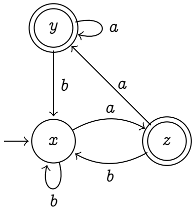
$reach(det(rev(M)))$:



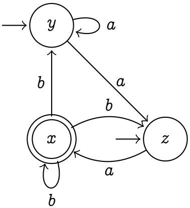
Brzozowski's algorithm: Example

[Adapted from slides by Jan Rutten (2012)]

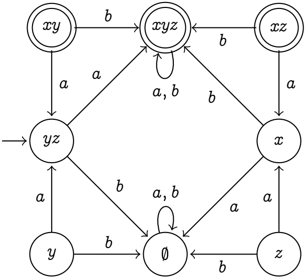
M:



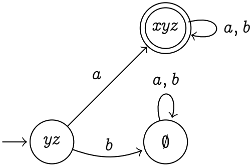
$rev(M)$:



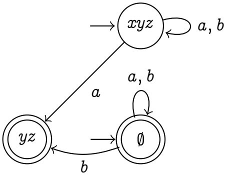
$det(rev(M))$:



$reach(det(rev(M)))$:



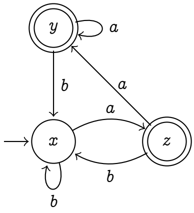
$rev(reach(det(rev(M))))$:



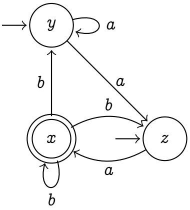
Brzozowski's algorithm: Example

[Adapted from slides by Jan Rutten (2012)]

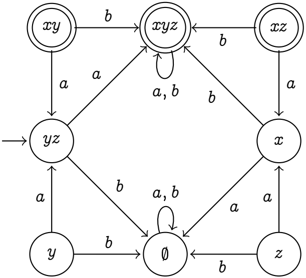
M:



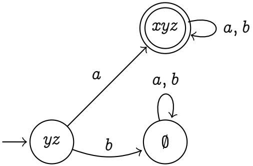
$rev(M)$:



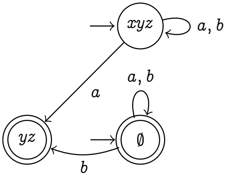
$det(rev(M))$:



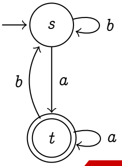
$reach(det(rev(M)))$:



$rev(reach(det(rev(M))))$:



$min(M)$:



Taking Stock

FSMs can be minimized by collapsing their states

- A characterization of $\approx$, a "collapsing" equivalence on Q
- The quotient FSM $M/_\approx$
- An algorithm for computing $\approx$
- Briefly: Brzozowski's algorithm

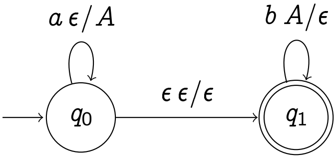
This concludes our study of regular languages (and their machines)

Next Lecture

- Context-free languages, revisited
- Pushdown machines

A Pushdown Machine

Accepting the (non-regular language) $L_1 = \{a^n b^n \mid n \geq 1\}$:



- $a\ X / Y$ means: a is read, and in the stack symbol X is replaced by symbol Y (symbols a, X, Y can be ϵ)