Elementary Computation Theory

Ming-Hsien Tsai

Institute of Information Science
Academia Sinica

FLOLAC 2017

Outline

- Finite state automata
- Regular Expressions
- WS1S
- ω-Automata
- Linear temporal logic

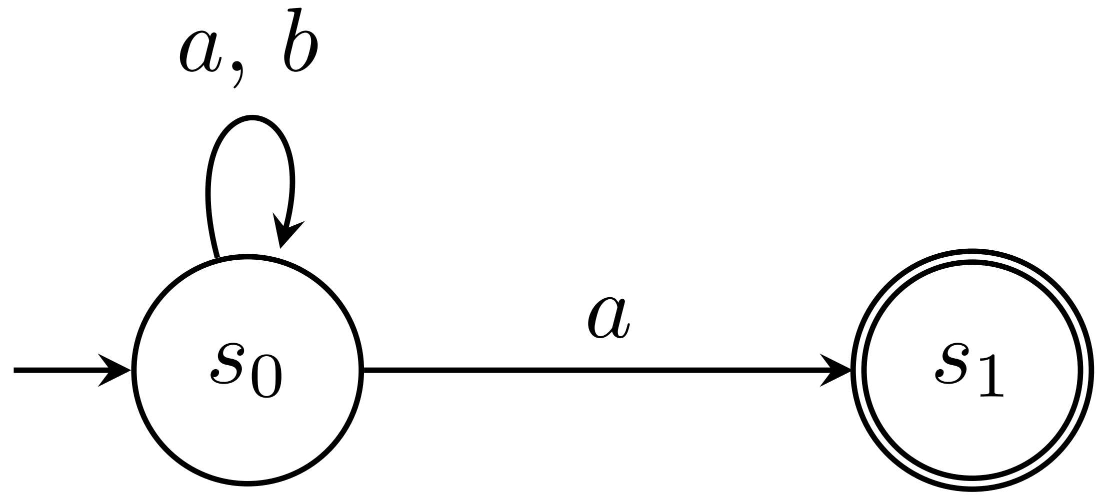
Computation

- What is the model of a computation machine?
- What is the result of a computation?

Computation

- What is the model of a computation machine?
- What is the result of a computation?
- The simplest model of computation machinery
  - *Finite state automata* (FSA), or equivalently nondeterministic finite automata (NFA), *nondeterministic finite word automata* (NFW)

Automaton M_1



- This automaton recognizes *words* (strings) end with an "a".

- Alphabet: $\{a,\ b\}$
- States: $\{s_0,\ s_1\}$
- Initial states: $\{s_0\}$
- Transitions: $\{(s_0,\ a,\ s_0),\ (s_0,\ a,\ s_1),\ (s_0,\ b,\ s_0)\}$
- Accepting states: $\{s_1\}$

Alphabet

- An *alphabet* is a set of symbols.
- Types of alphabet: *classical* and *propositional*
- Examples:
  - $\{a,\ b\}$
  - $\{send,\ receive,\ ack\}$
  - $\{(p\ \ q),\ (\neg p\ \ q),\ (p\ \neg q),\ (\neg p\ \neg q)\}$

Words

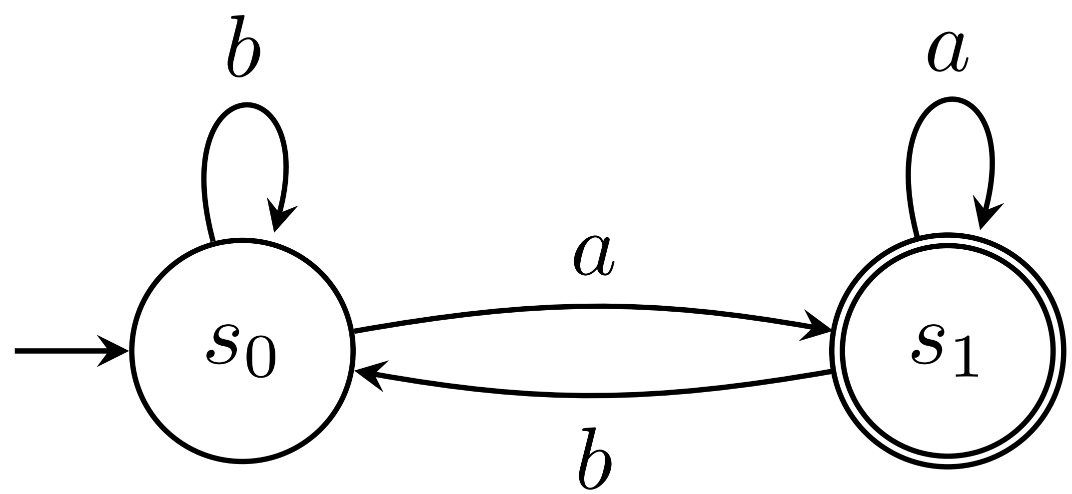
- Let Σ be a finite alphabet.
- A *word* w over Σ ($w \in \Sigma^*$) is a sequence of symbols $a_0 a_1 a_2 \ldots a_{n\text{-}1}$ with $a_i \in \Sigma$.
    - Length of w is n.
    - The empty word is denoted by ϵ.
- Examples ($\Sigma = \{a, b\}$):
    - $a\ b\ b\ a$
    - $a\ b\ a\ b\ a\ b$

w^* : repeat w finitely many times

Finite State Automata Syntax

- A finite state automaton is a 5-tuple $(Q, \Sigma, \delta, I, F)$ where
  - Q is a finite set of *states*,
  - Σ is a finite *alphabet*,
  - $\delta : Q \times \Sigma \to 2^Q$ is the *transition function* (sometimes written as a relation $\delta : Q \times \Sigma \times Q$),
  - $I \subseteq Q$ is the set of *initial states*, and
  - $F \subseteq Q$ is the set of *accepting (final) states*

Automaton M_2



$$A = (Q,\ \Sigma,\ \delta,\ I,\ F)$$

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

$Q = ?$

$I = ?$

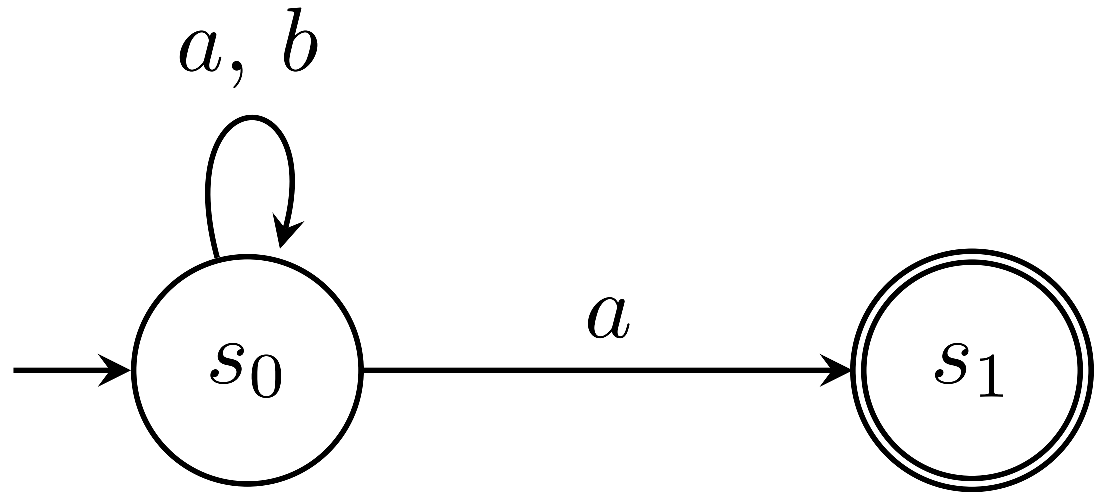
$\delta = ?$

$F = ?$

Finite State Automata Semantics

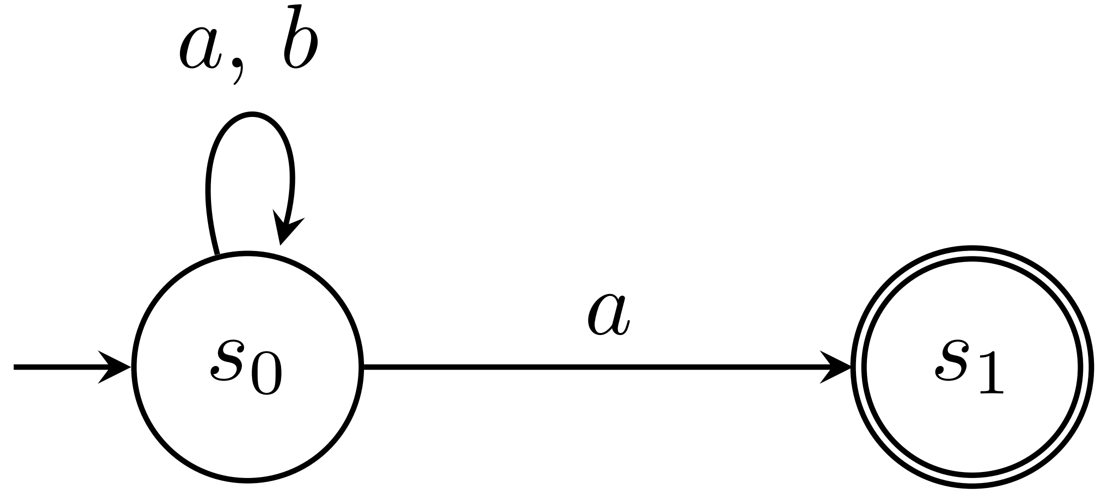
- Let $M = (Q,\ \Sigma,\ \delta,\ I,\ F)$ be a finite state automaton.
- Let $w = \ a_0 a_1 a_2 \ldots a_{n\text{-}1}$ be a word over Σ.
- A *run* of w on M is a sequence of states $s_0 s_1 s_2 \ldots s_n$ where
  - $s_0 \in I$
  - $(s_i,\ a_i,\ s_{i+1}) \in \delta$

Runs



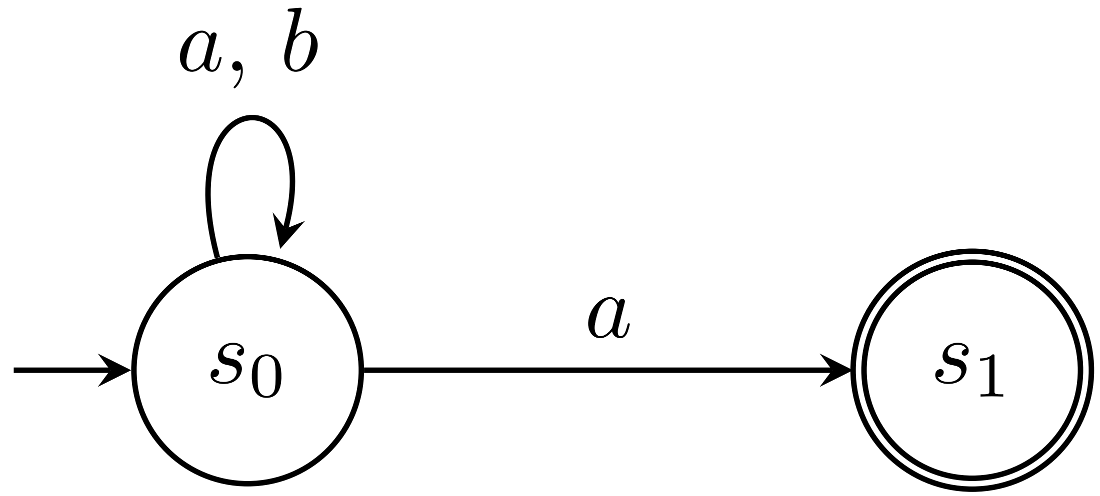
- What are the runs of the following words?
  - $a\ b\ a\ b$
  - $a\ b\ b\ a$

Runs



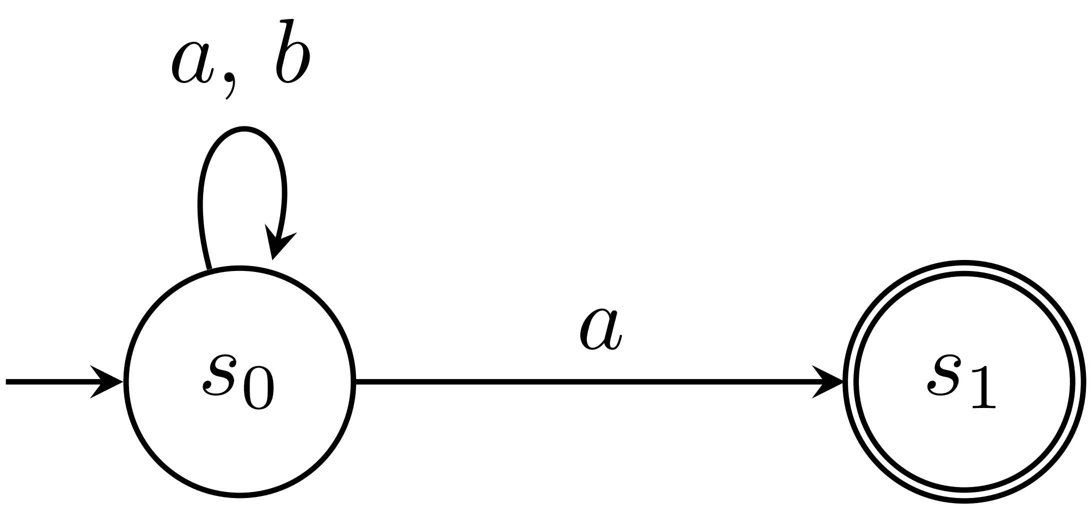
- What are the runs of the following words?
  - $a\ b\ a\ b$ $\qquad s_0\ s_0\ s_0\ s_0\ s_0$
  - $a\ b\ b\ a$

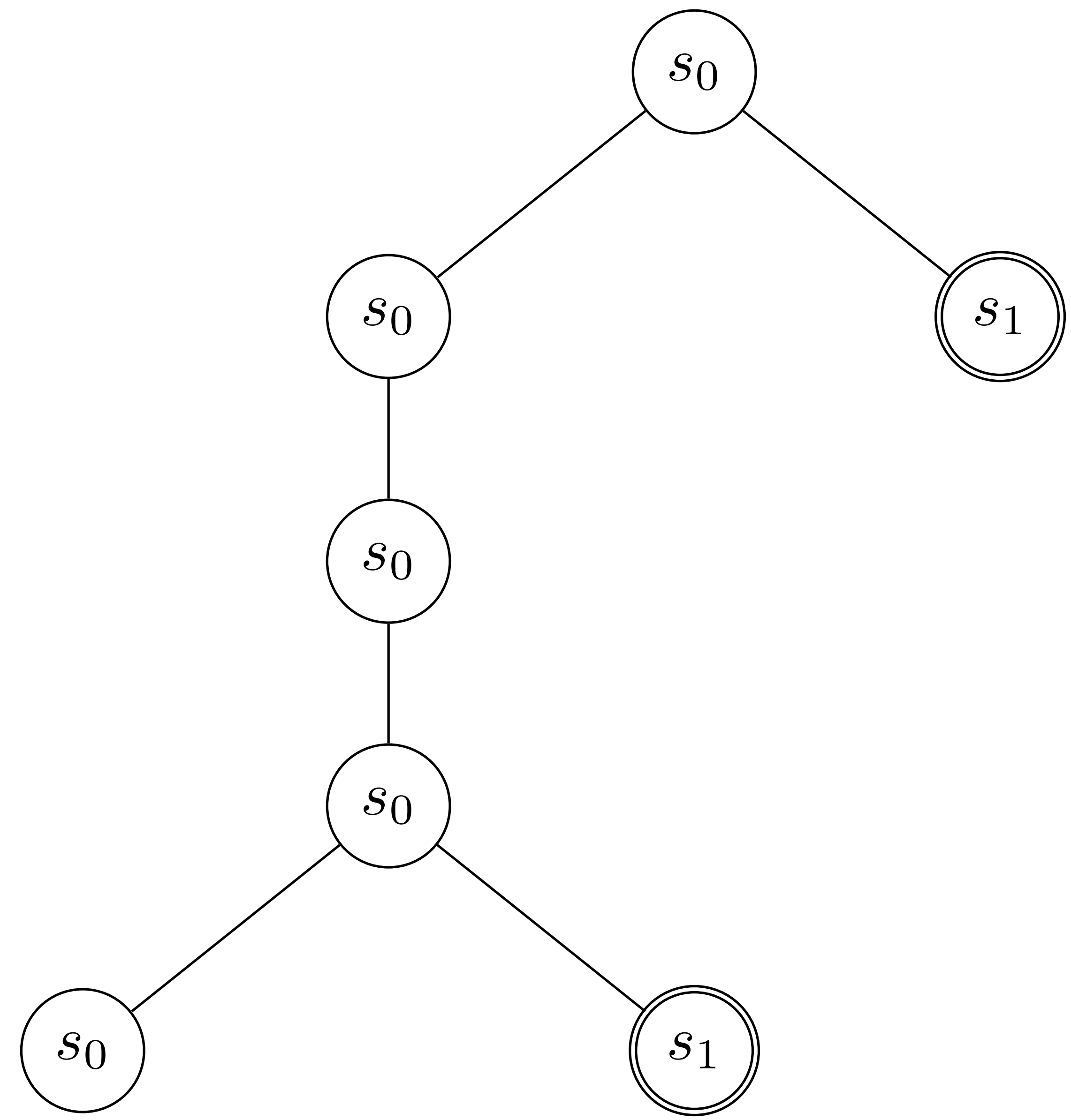
Runs



- What are the runs of the following words?
  - $a\ b\ a\ b$ $\qquad s_0\ s_0\ s_0\ s_0\ s_0$
  - $a\ b\ b\ a$ $\qquad s_0\ s_0\ s_0\ s_0\ s_0$ and $s_0\ s_0\ s_0\ s_0\ s_1$

Run Tree



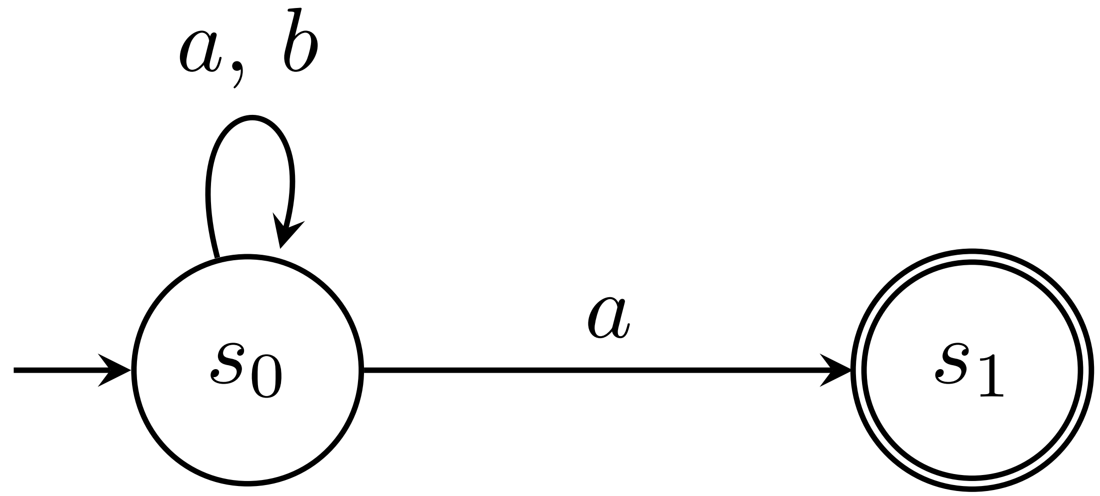


the run tree of $abba$ on M_1

Finite State Automata Semantics (cont'd)

- $M = (Q,\ \Sigma,\ \delta,\ I,\ F)$
- A run $s_0 s_1 s_2 \ldots s_n$ is *accepting* if $s_n \in F$.
- A word w is accepted by M if there is an accepting run of w on M.
- The *language* of M is the set of strings accepted by M, denoted by $L(M)$.

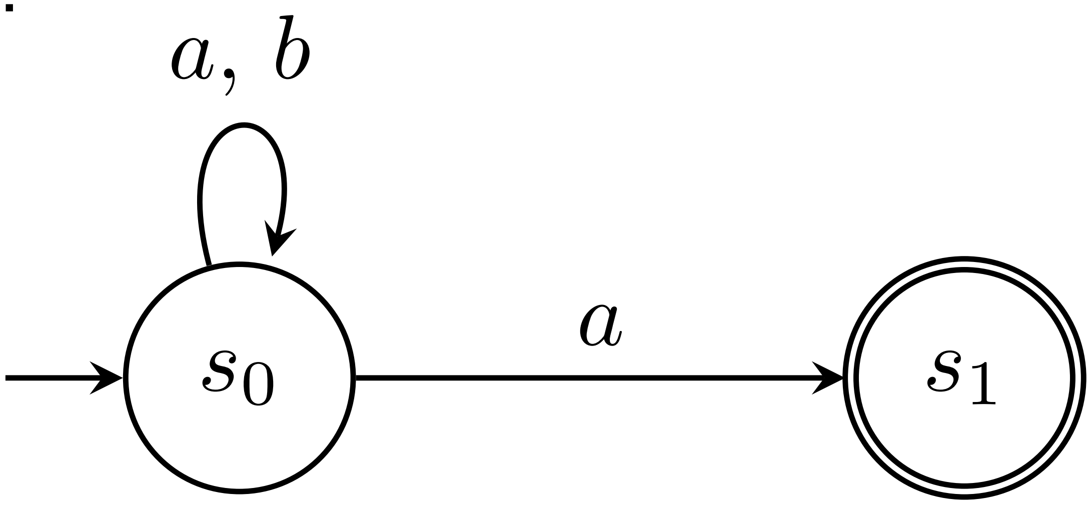
Accepting Runs



- Which run is accepting?
  - $s_0\ s_0\ s_0\ s_0\ s_0$
  - $s_0\ s_0\ s_0\ s_0\ s_1$

Languages

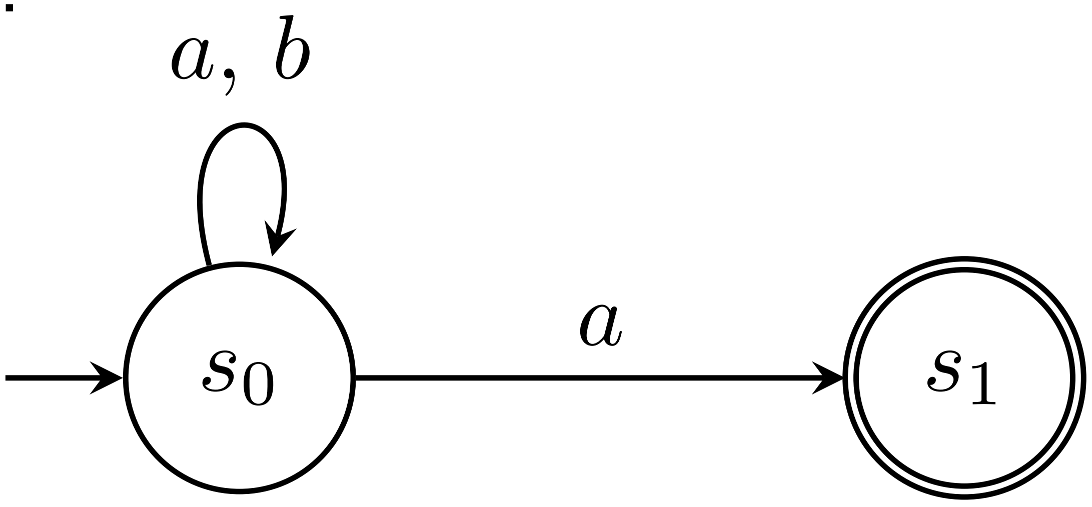
- What is the language of M_1?



- The language recognized by a finite state automaton is a *regular language*.

Languages

- What is the language of M_1?



$L(M_1) = \{\ a_0a_1\ldots a_n \mid n\in\mathbb{N}$ *and* $a_n = a\}$

- The language recognized by a finite state automaton is a *regular language*.

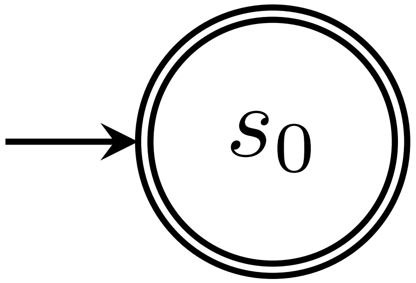
Exercise

- Given an alphabet $\{1, 2, +\}$, draw a finite state automaton such that the automaton accepts words evaluated to 3.

Emptiness and Universality

- $M = (Q, \Sigma, \delta, I, F)$
- An automaton M is *empty* if $L(M) = \varnothing$.
- An automaton M is *universal* if $L(M) = \Sigma^*$.

Emptiness and Universality

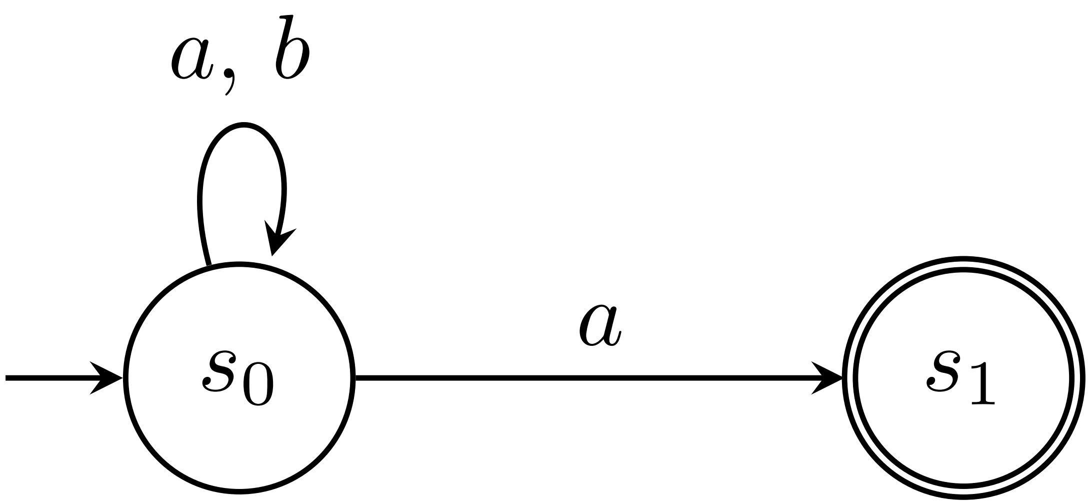


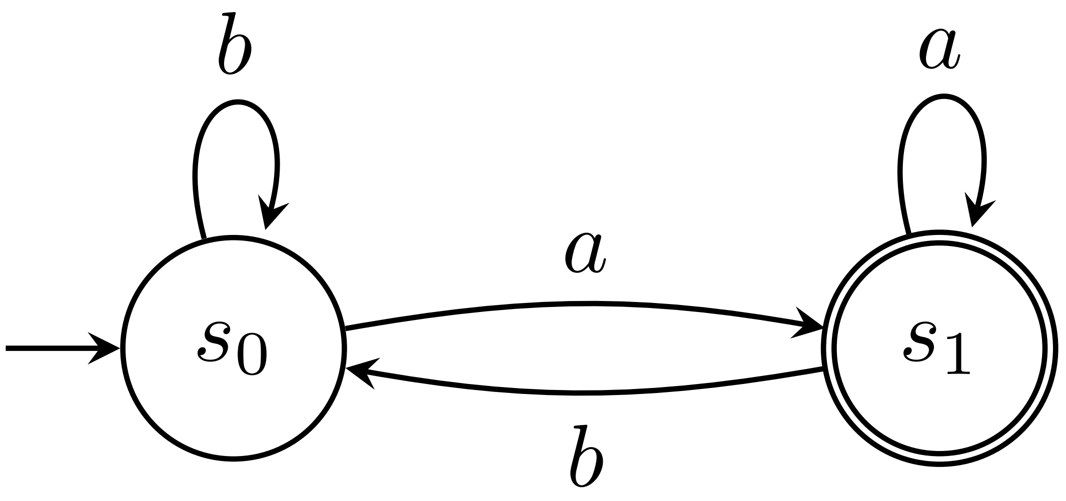
is this automaton empty?

- $M = (Q, \Sigma, \delta, I, F)$
- An automaton M is *empty* if $L(M) = \varnothing$.
- An automaton M is *universal* if $L(M) = \Sigma^*$.

Equivalence

- Two automata are *equivalent* if they recognize the same language.





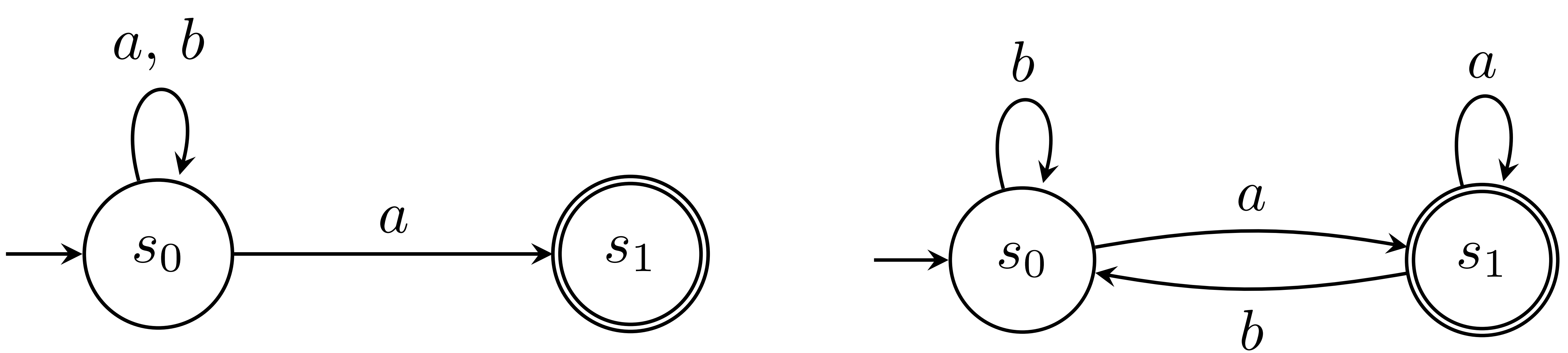
$$L(M_1) = L(M_2)?$$

Deterministic Finite Automata (DFA)

- An automaton $M = (Q, \Sigma, \delta, I, F)$ is *deterministic* if
  - $|I| = 1$, and
  - $|\delta(s, a)| = 1$ for all $s \in Q$ and $a \in \Sigma$.

(is complete if $|\delta(s, a)| \geq 1$)

- Which one is deterministic?



Determinism VS Nondeterminism

- Let D be a DFA. The language $L(D)$ is accepted by the NFA D. (A DFA is also an NFA.)
- Let N be an NFA. Can we construct a DFA D such that $L(D) = L(N)$?

Determinism VS Nondeterminism

- Let D be a DFA. The language $L(D)$ is accepted by the NFA D. (A DFA is also an NFA.)

- Let N be an NFA. Can we construct a DFA D such that $L(D) = L(N)$?

Determinism VS Nondeterminism

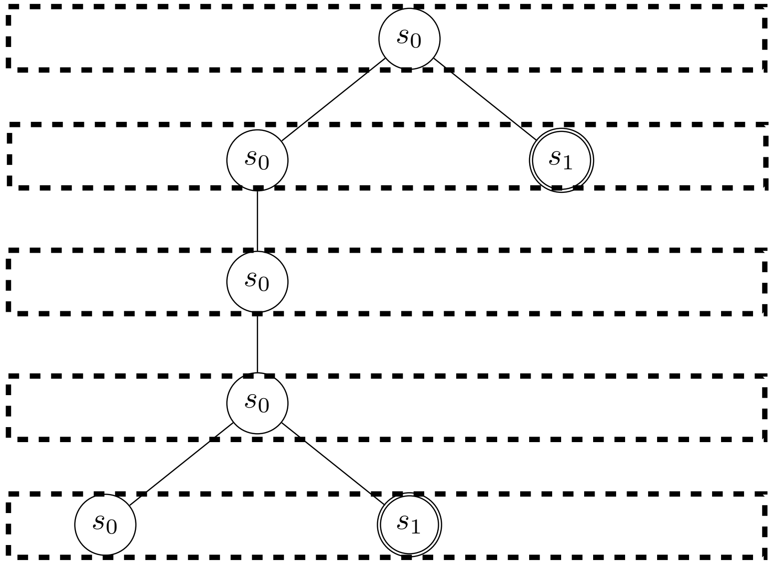
- Let D be a DFA. The language $L(D)$ is accepted by the NFA D. (A DFA is also an NFA.)

- Let N be an NFA. Can we construct a DFA D such that $L(D) = L(N)$?

- DFA and NFA have the same expressive power.

Determinization

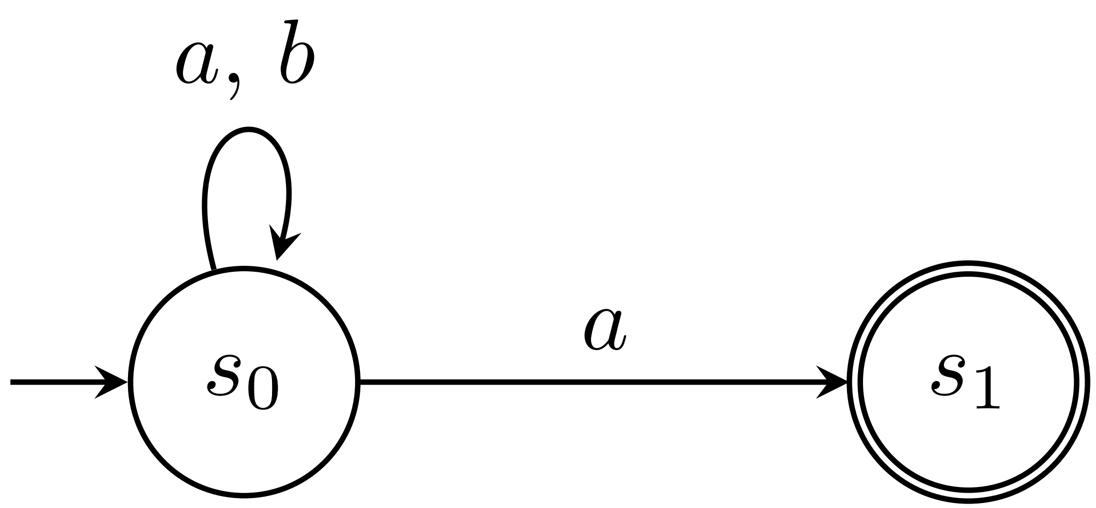
- Let $N = (Q, \Sigma, \delta, I, F)$.
- By *subset construction*, define $D = (2^Q, \Sigma, \Delta, \{ I \}, G)$ where
  - $\Delta(S, a) = \cup_{s \in S} \delta(s, a)$, and
  - $G = \{ S \in 2^Q \mid S \cap F \neq \emptyset \}$.

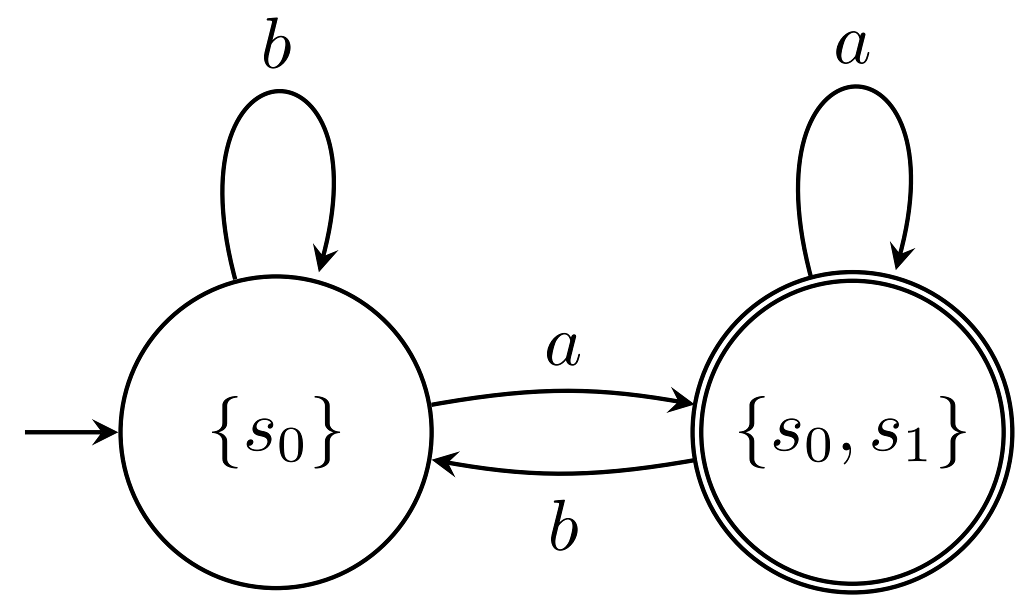


- We can show that $L(N) = L(D)$ by induction on the length of input words.

Subset Construction

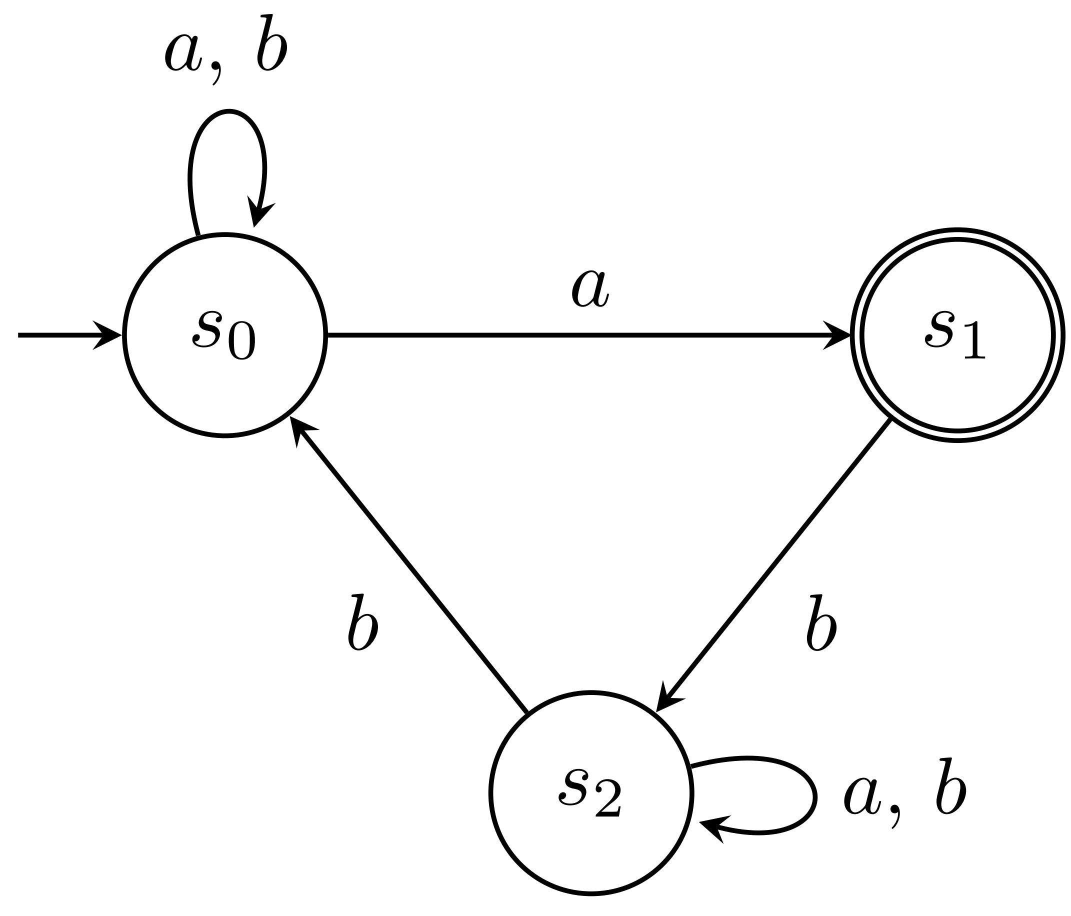
- What is the determinization of M_1?





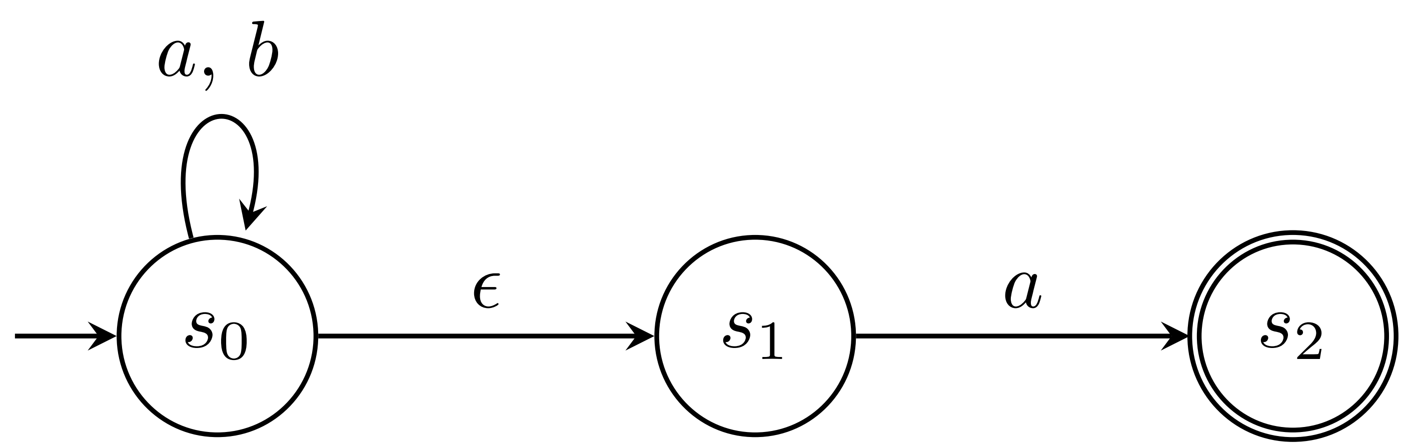
Exercise

- Apply subset construction to determinize the following automaton



ϵ-Transitions

- Assume ϵ does not belong to the alphabet.
- An ϵ-transition is a transition that does not need to consume any symbol.
- ϵ-transitions are only allowed in NFA.
- DFA and NFA with ϵ-transitions have the same expressive power.

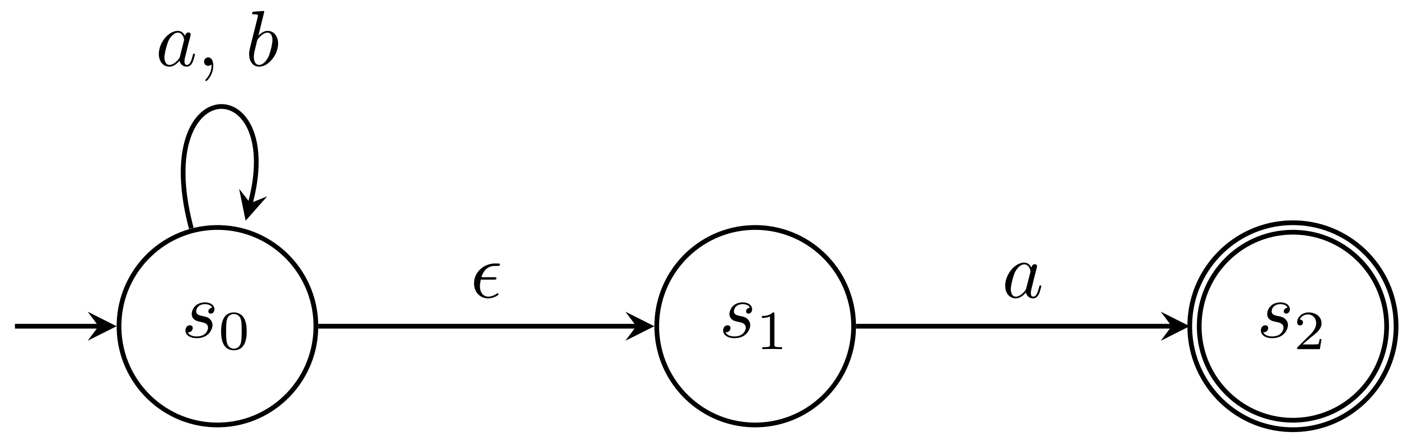


Elimination of ϵ-Transitions

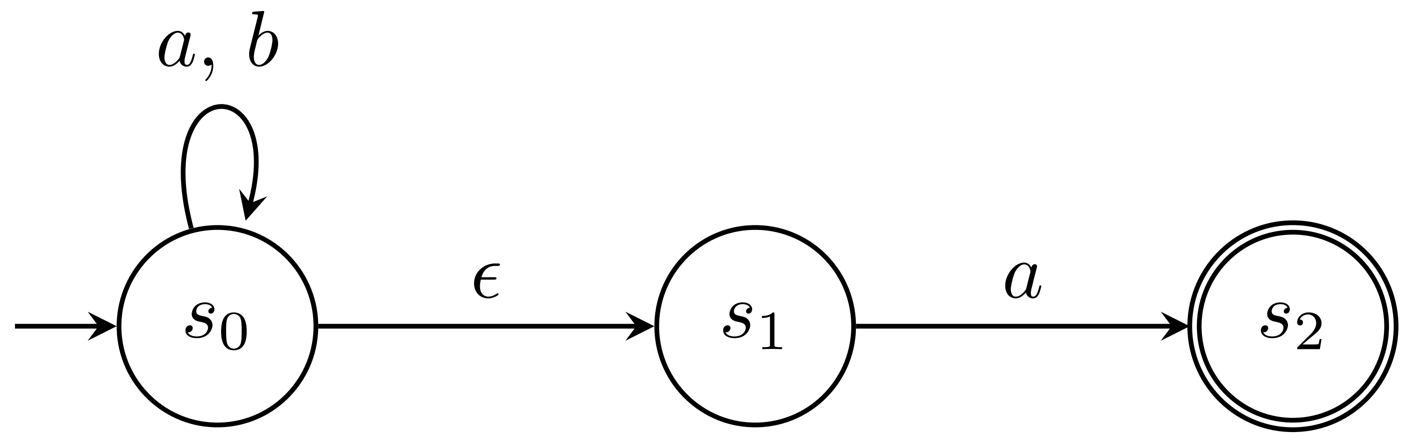
- $M = (Q, \Sigma \cup \{\epsilon\}, \delta, I, F)$ is an NFA with ϵ-transitions.
- Let $E(X)$ denote the ϵ-closure of $X \subseteq Q$.
  - $E(X) = \{ s \mid s \in X$ *or* s *is reachable from a state in* X *through* ϵ-*transitions* $\}$
- Construct an NFA $N = (Q, \Sigma, \boldsymbol{\Delta}, J, F)$ *where*
  - $\boldsymbol{\Delta}(s, a) = E(\delta(s, a))$, and
  - $J = E(I)$

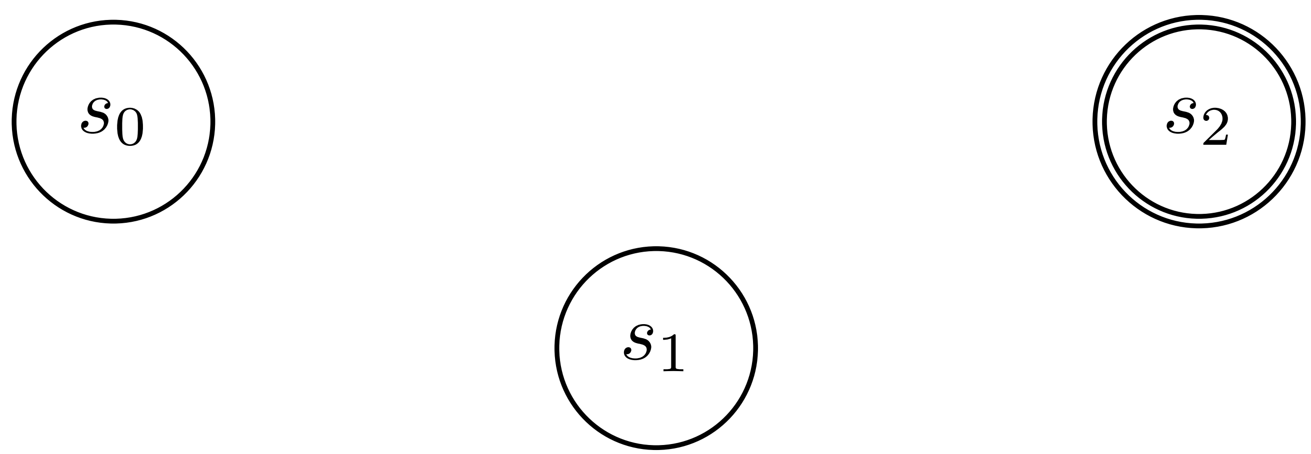
Elimination of ϵ-Transitions

Example



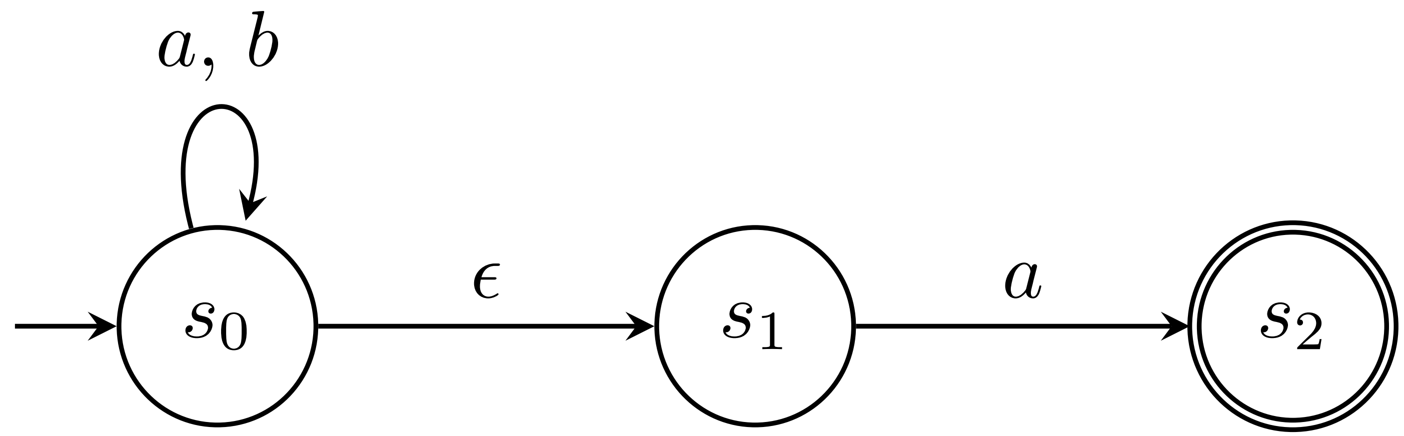
Elimination of ϵ-Transitions Example

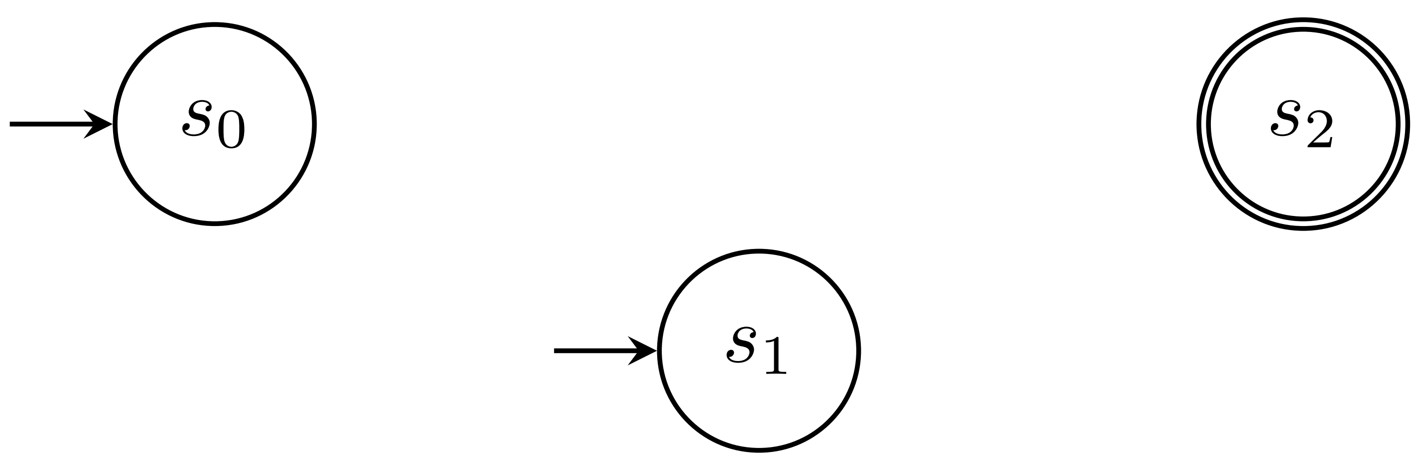




Elimination of ϵ-Transitions

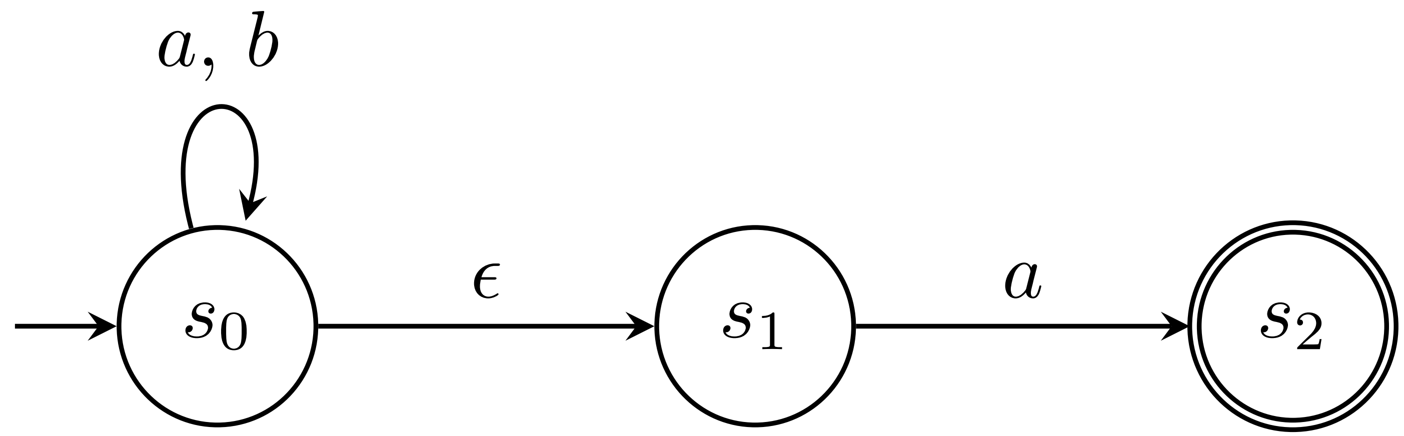
Example

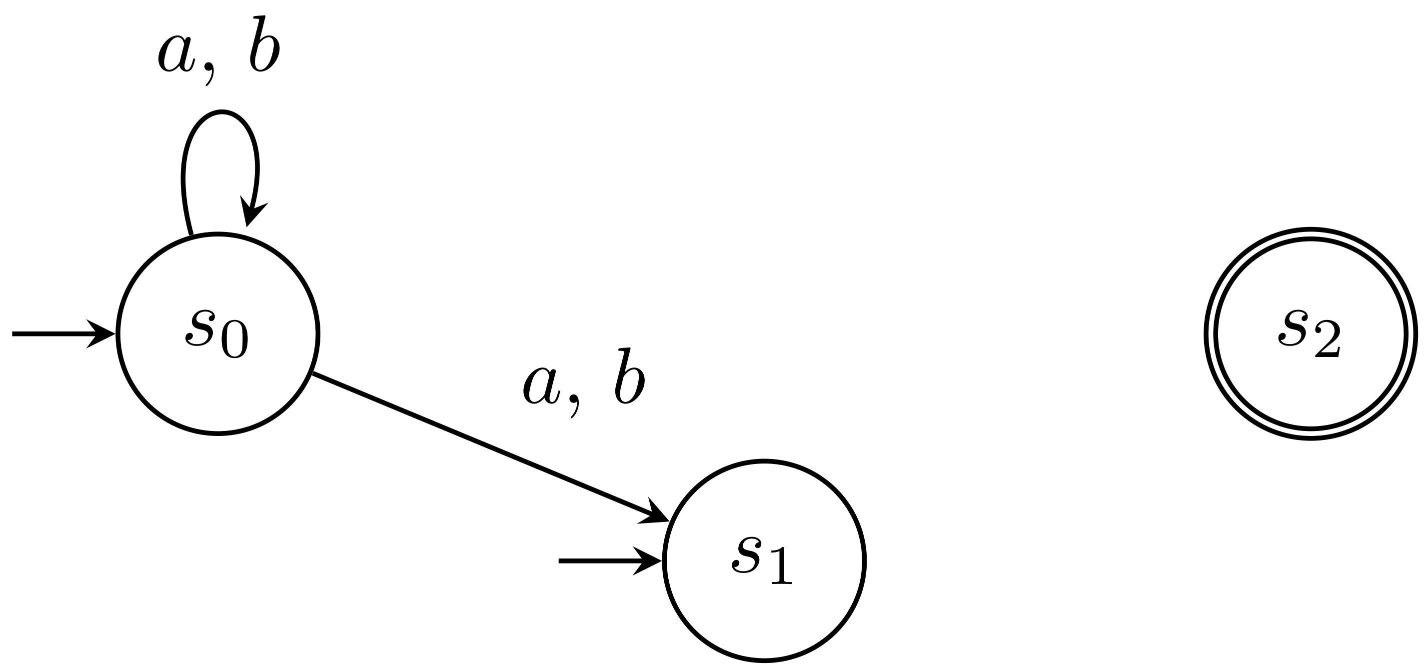




Elimination of ϵ-Transitions

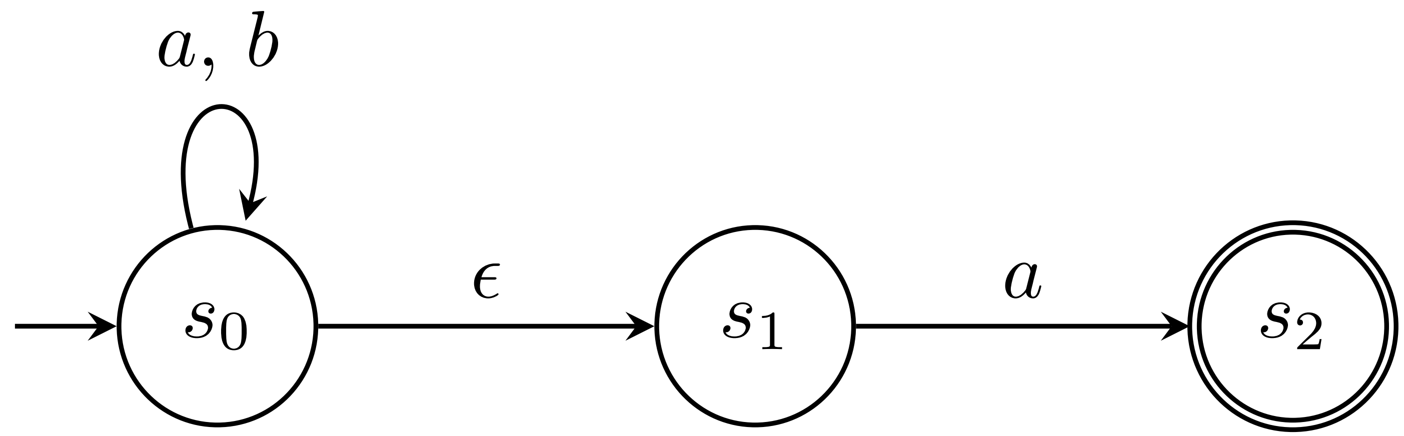
Example

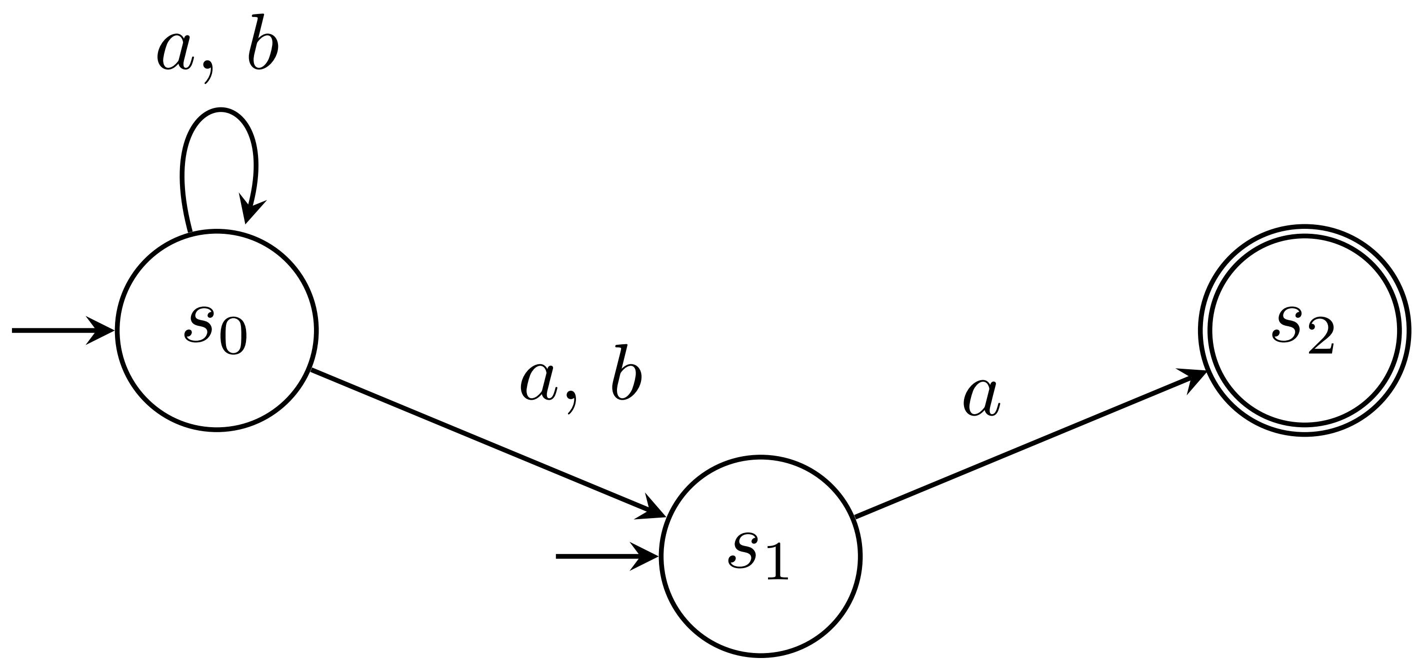




Elimination of ϵ-Transitions

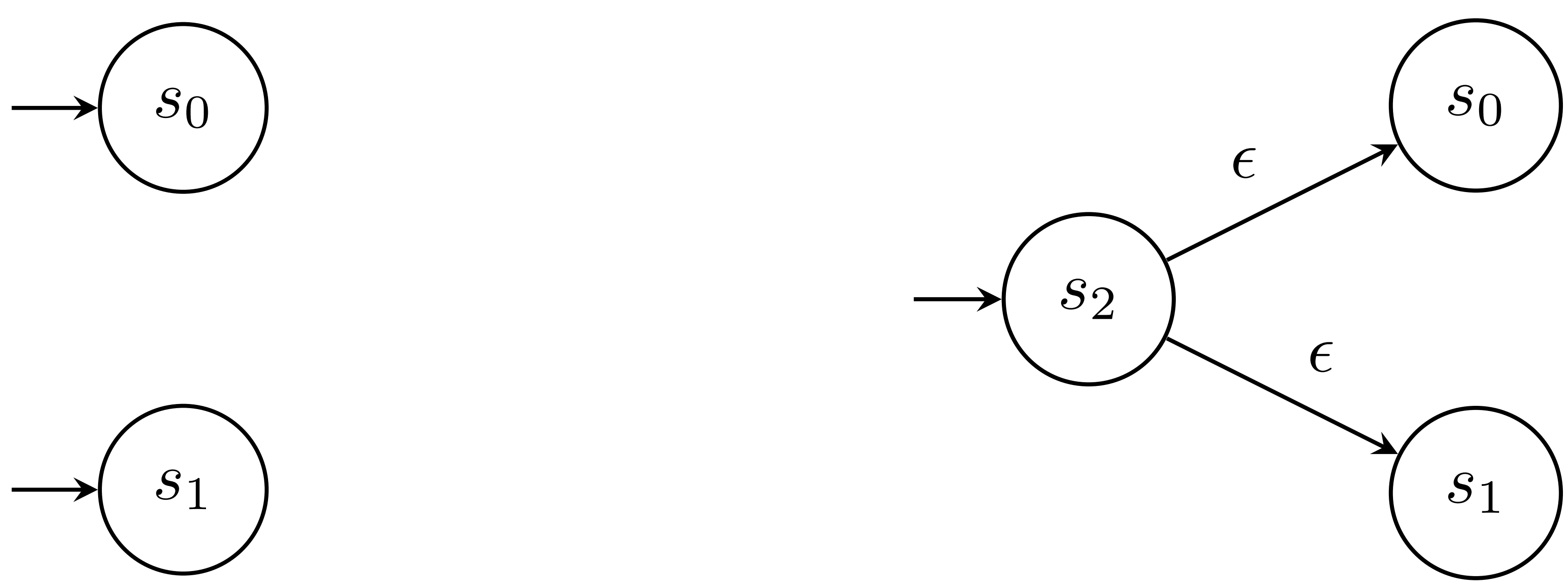
Example





Single Initial State

- NFA may be defined as automata with single initial state.
- NFA with multiple initial states does not have more expressive power.



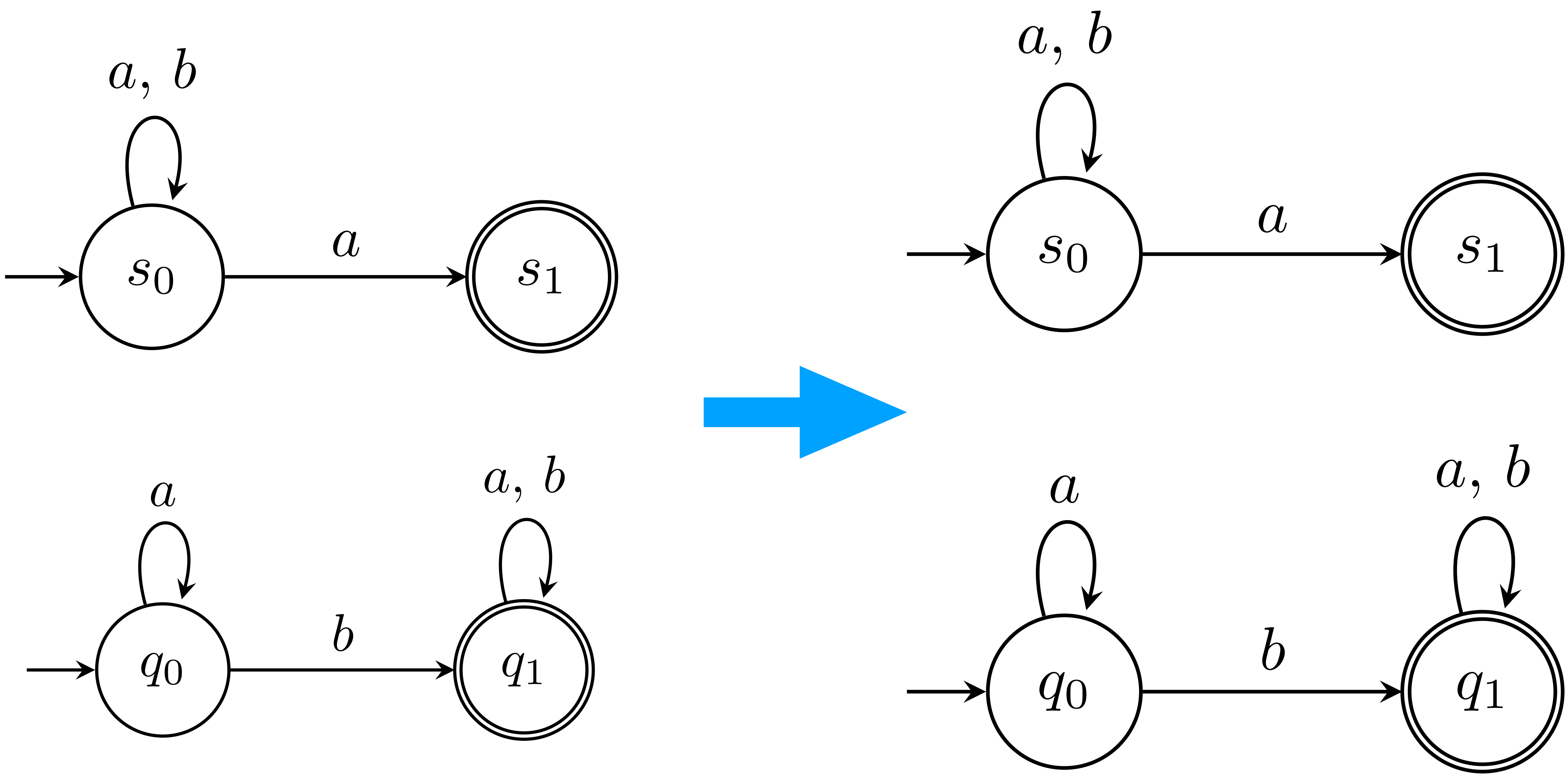
Closure Properties

- Regular languages are closed under the following operations.
  - union,
  - intersection,
  - concatenation,
  - Kleene closure, and
  - complementation.

Union

- $M_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$
- *Assume* $Q_1 \cap Q_2 = \emptyset$.
- $M_3 = (Q_1 \cup Q_2, \Sigma, \delta_3, I_1 \cup I_2, F_1 \cup F_2)$ where $(s, a, t) \in \delta_3$ if
  - $(s, a, t) \in \delta_1$, *or*
  - $(s, a, t) \in \delta_2$
- $L(M_3) = L(M_1) \cup L(M_2)$

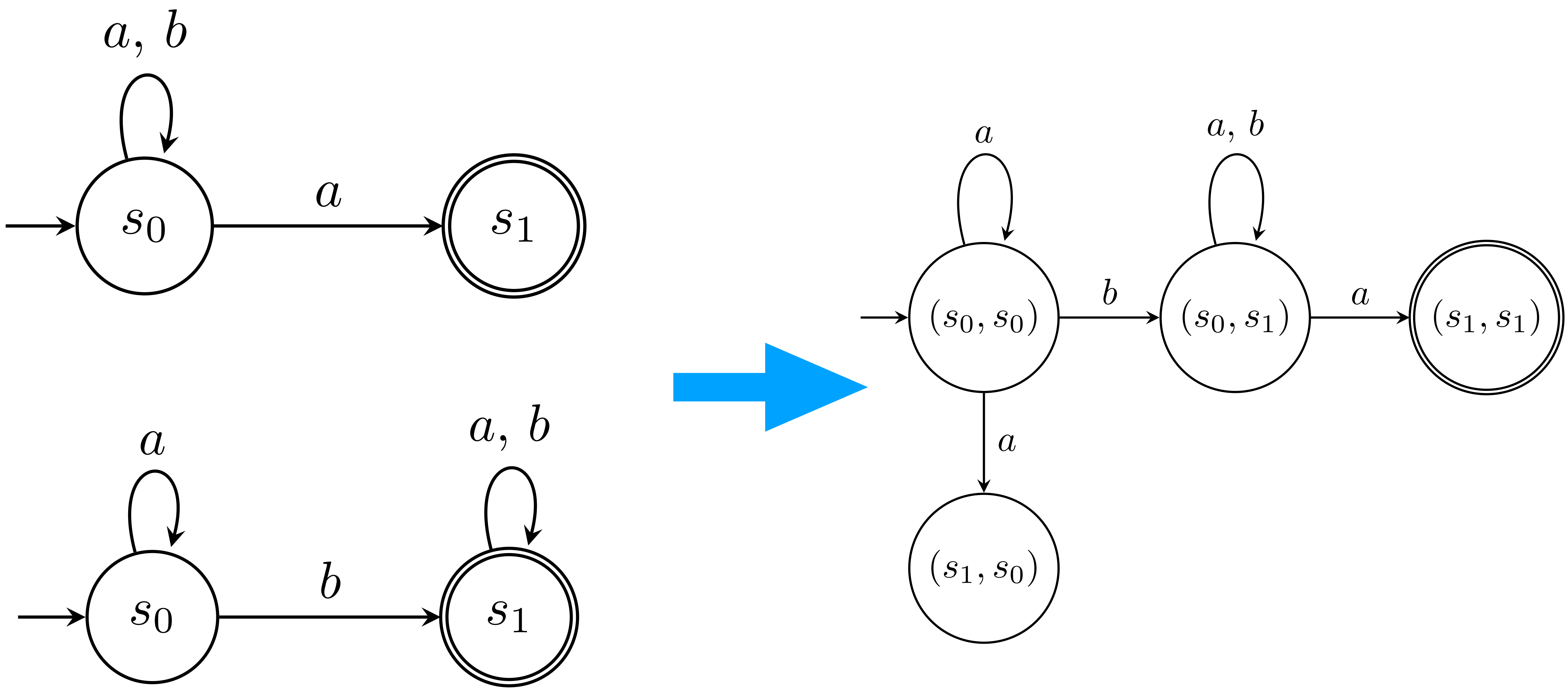
Union
Example



Intersection

- $M_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$
- $M_3 = (Q_1 \times Q_2, \Sigma, \delta_3, I_1 \times I_2, F_1 \times F_2)$ where $((s_1, s_2), a, (t_1, t_2)) \in \delta_3$ if
  - $(s_1, a, t_1) \in \delta_1$, and
  - $(s_2, a, t_2) \in \delta_2$
- $L(M_3) = L(M_1) \cap L(M_2)$

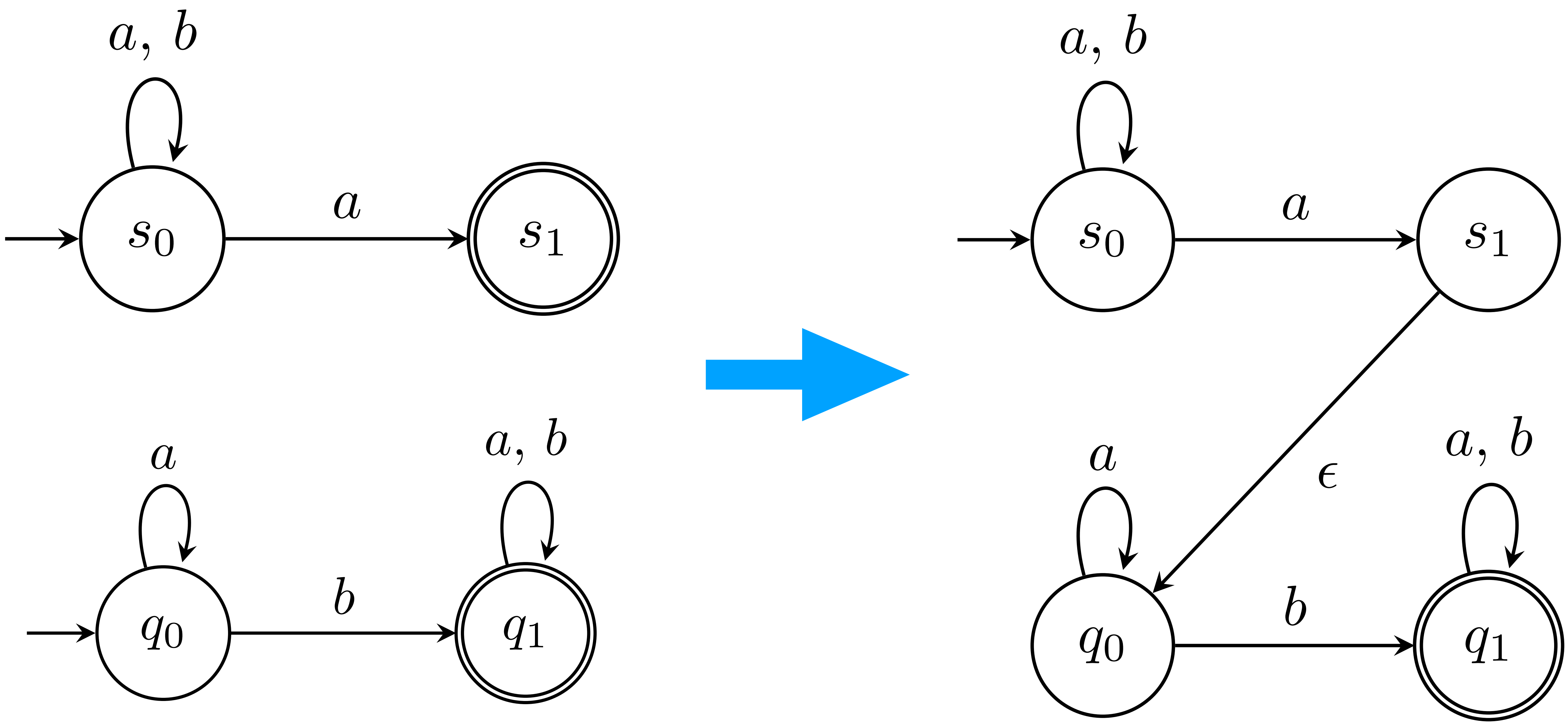
Intersection Example



Concatenation

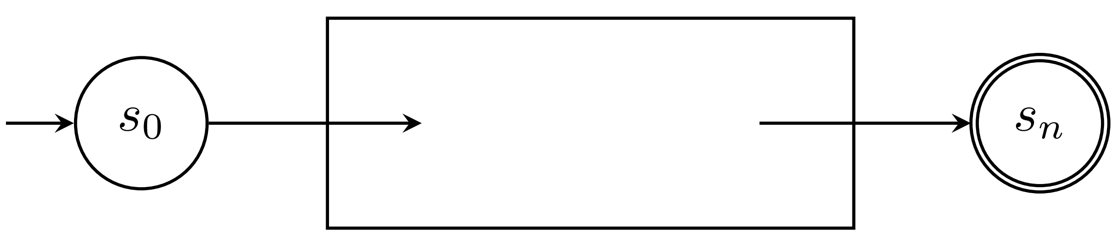
- $M_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$
- *Assume* $Q_1 \cap Q_2 = \varnothing$ *and* $\epsilon \notin \Sigma$.
- $M_3 = (Q_1 \cup Q_2, \Sigma \cup \{\epsilon\}, \delta_3, I_1, F_2)$ where $(s, a, t) \in \delta_3$ if
  - $(s, a, t) \in \delta_1$,
  - $(s, a, t) \in \delta_2$, *or*
  - $a = \epsilon$, $s \in F_1$, and $t \in I_2$.
- $L(M_3) = L(M_1)L(M_2) = \{ uv \mid u \in L(M_1) \textit{ and } v \in L(M_2) \}$

Concatenation Example



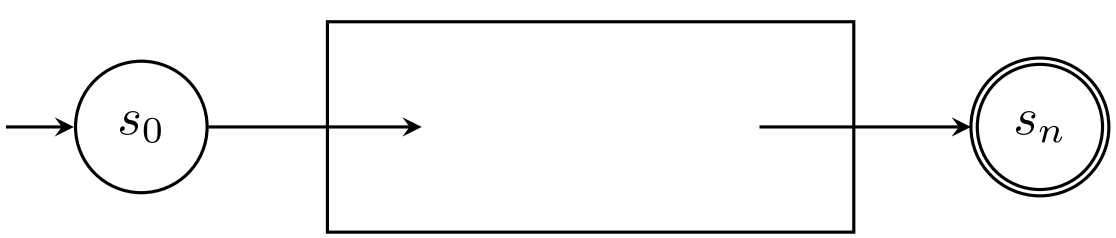
Kleene Closure

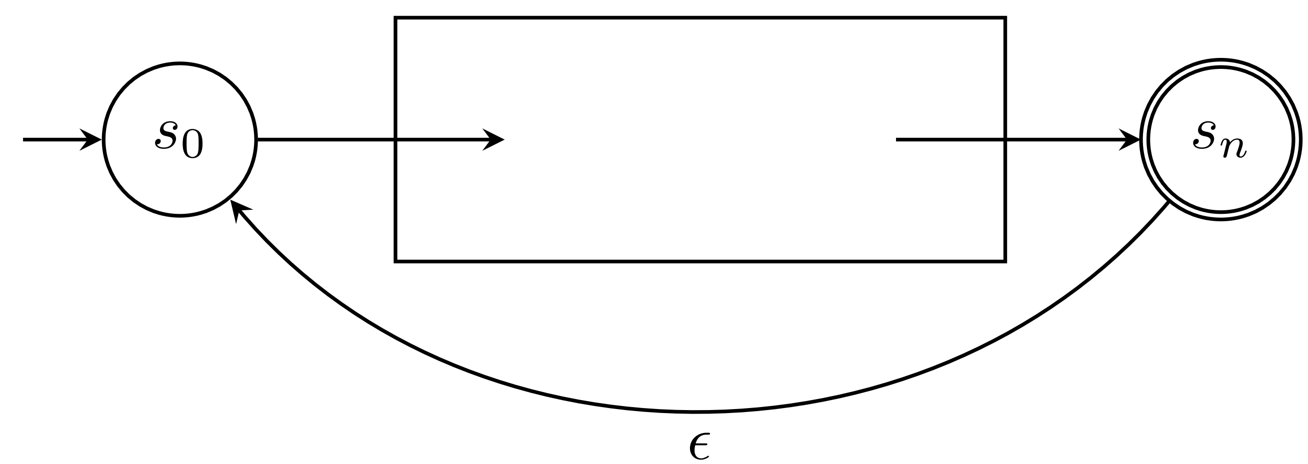
- An operation that repeat a string arbitrary number of times (including zero time).



Kleene Closure

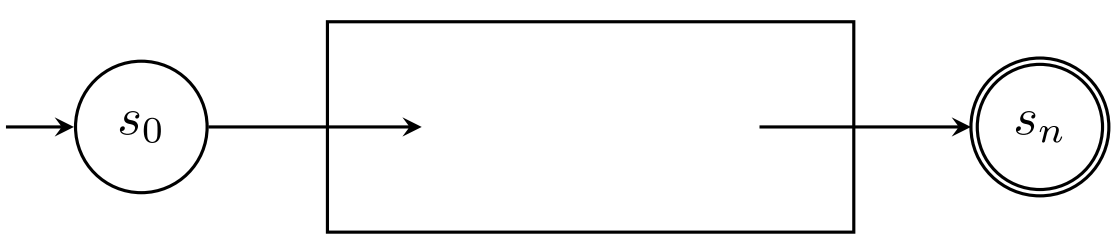
- An operation that repeat a string arbitrary number of times (including zero time).

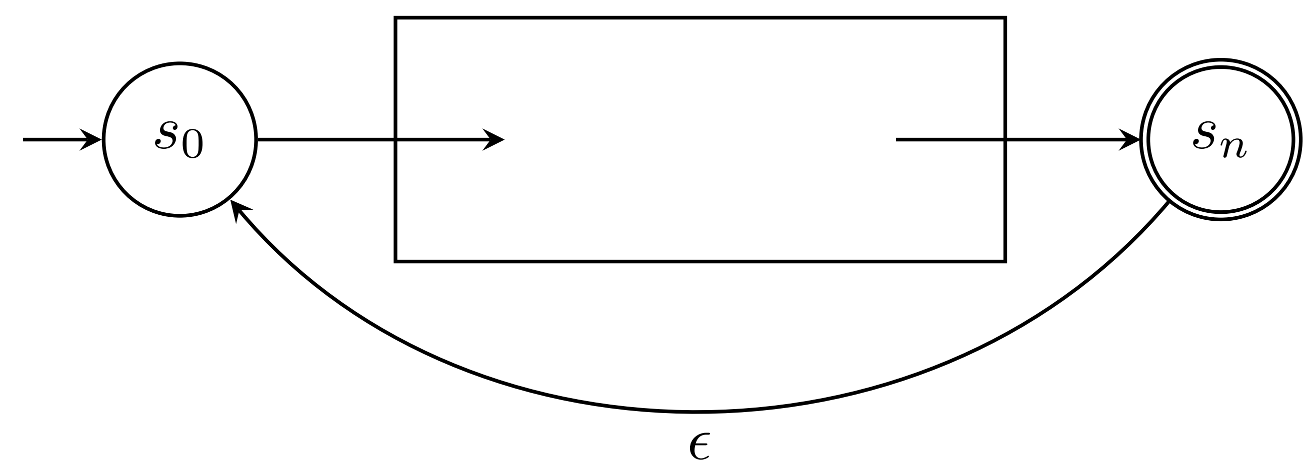


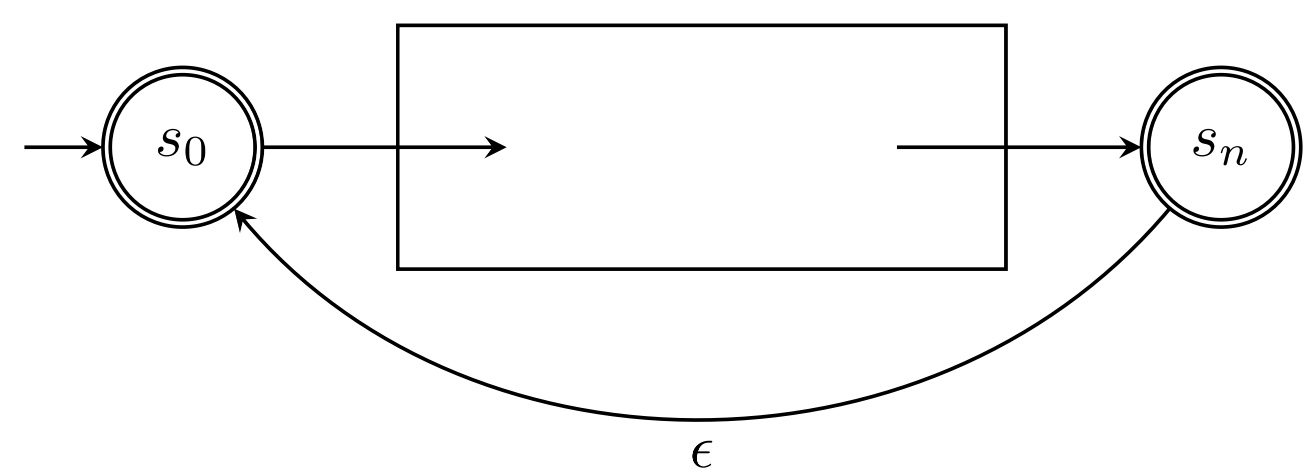


Kleene Closure

- An operation that repeat a string arbitrary number of times (including zero time).



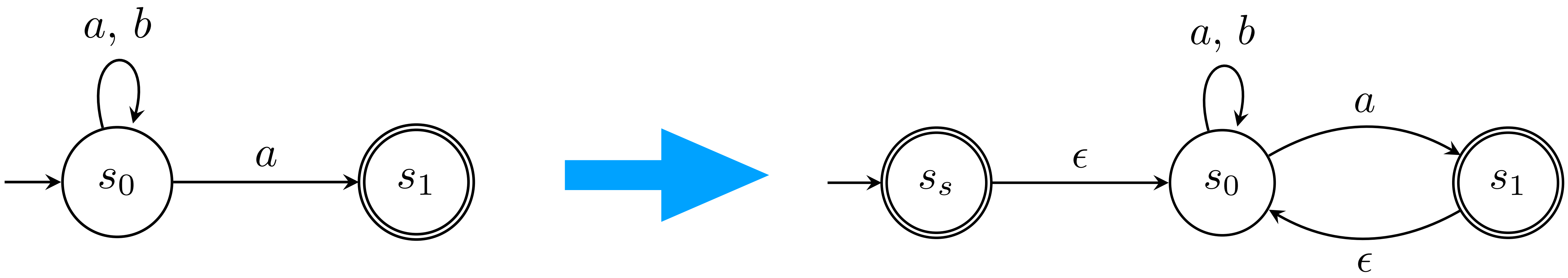




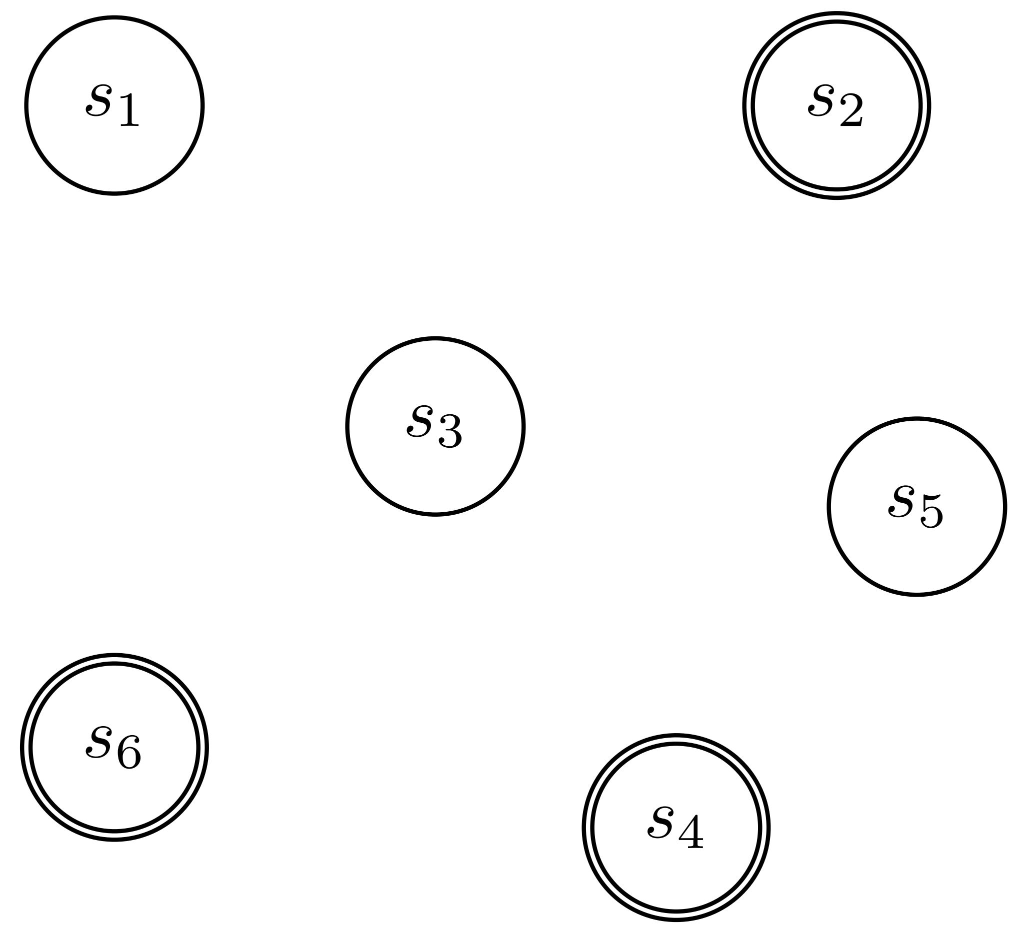
Kleene Closure (cont'd)

- $M = (Q, \Sigma, \delta, I, F)$
- Assume $\epsilon \notin \Sigma$ and $s_s \notin Q$.
- $M' = (Q \cup \{s_s\}, \Sigma \cup \{\epsilon\}, \Delta, \{s_s\}, F \cup \{s_s\})$ where $(s, a, t) \in \Delta$ if
  - $s = s_s$, $t \in I$, and $a = \epsilon$,
  - $(s, a, t) \in \delta$, or
  - $s \in F$, $t \in I$, and $a = \epsilon$.
- $L(M') = L(M)^*$

Kleene Closure Example



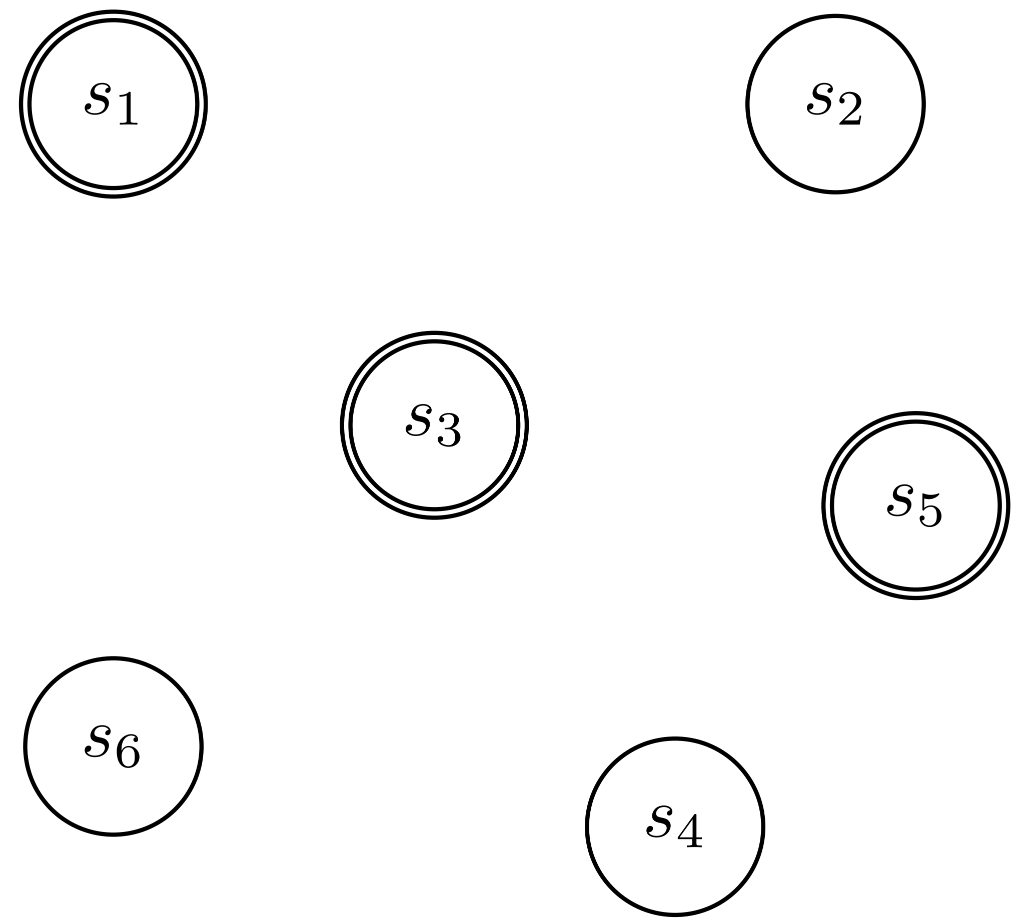
Complementation
DFA



- $M = (Q, \Sigma, \delta, I, F)$ is a DFA.
- $M' = (Q, \Sigma, \delta, I, Q \setminus F)$
- $L(M') = \Sigma^* \setminus L(M)$

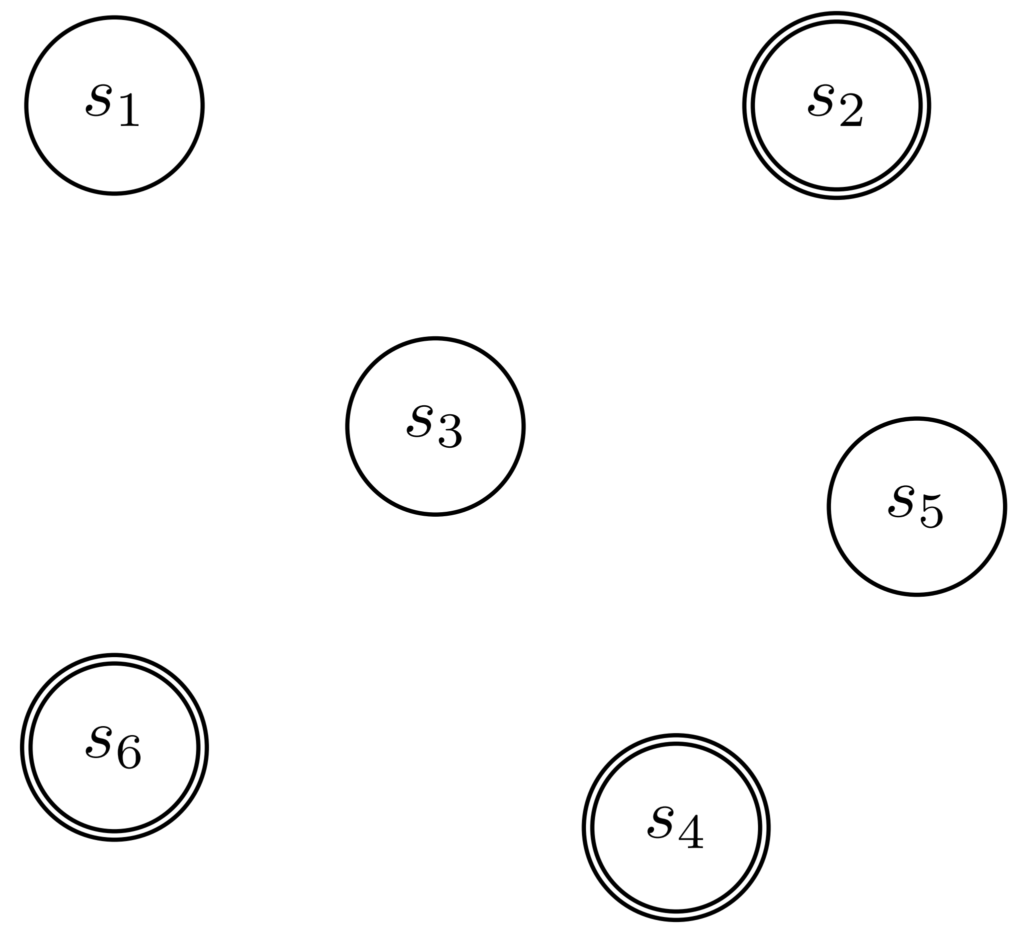
Complementation DFA

- $M = (Q, \Sigma, \delta, I, F)$ is a DFA.
- $M' = (Q, \Sigma, \delta, I, Q \setminus F)$
- $L(M') = \Sigma^* \setminus L(M)$



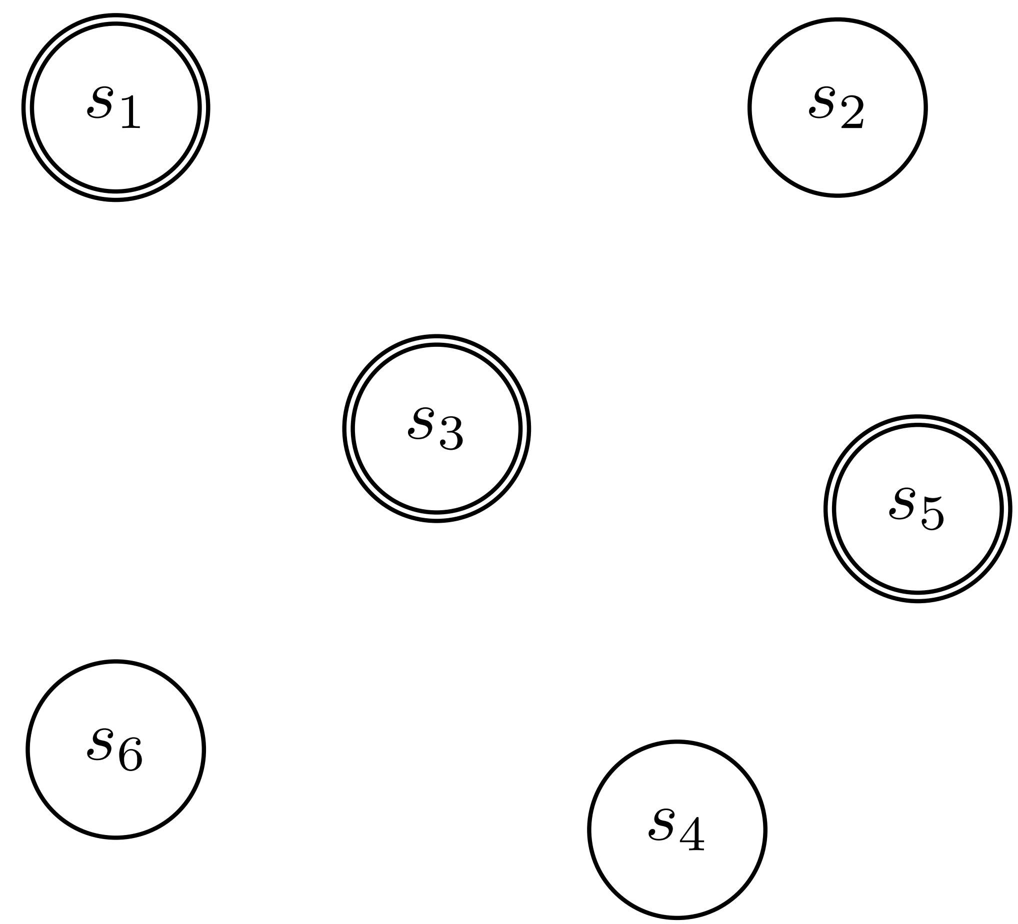
Complementation
NFA

- $M = (Q, \Sigma, \delta, I, F)$ is an NFA.
- $M' = (Q, \Sigma, \delta, I, Q \setminus F)$
- $L(M') = \Sigma^* \setminus L(M)$?



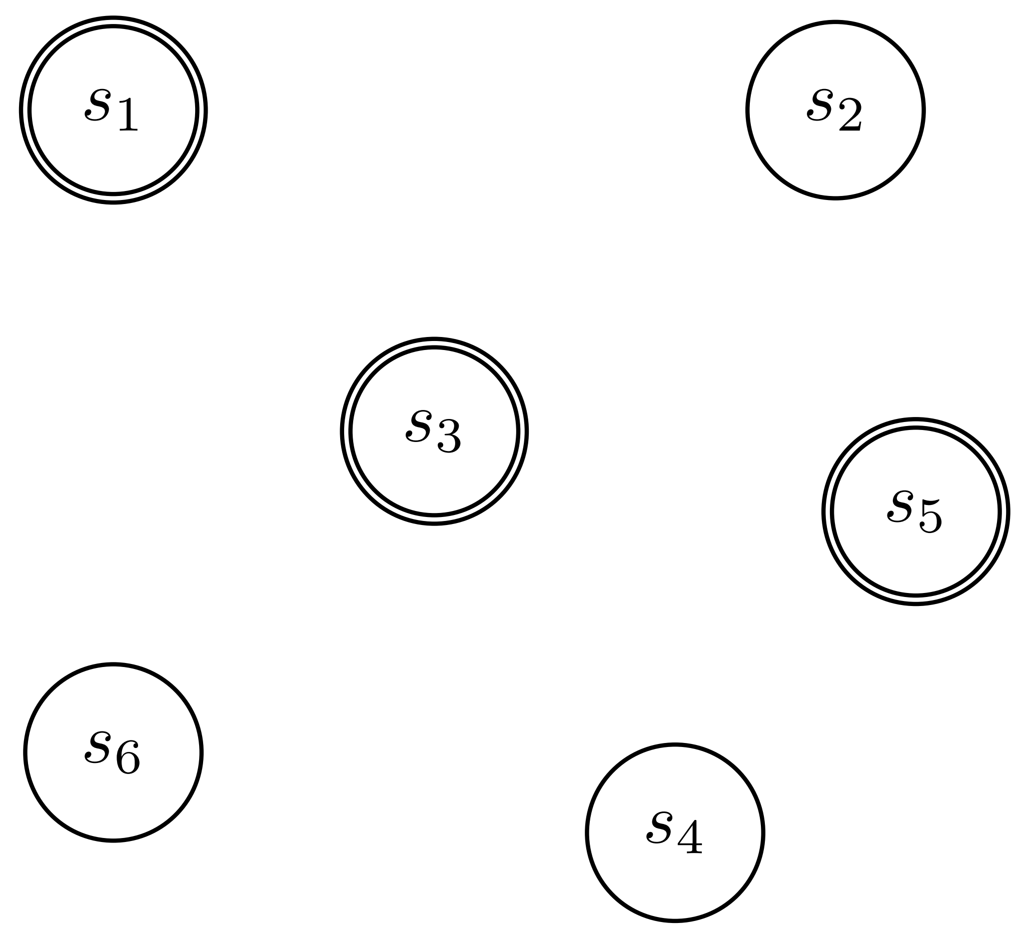
Complementation
NFA

- $M = (Q, \Sigma, \delta, I, F)$ is an NFA.
- $M' = (Q, \Sigma, \delta, I, Q \setminus F)$
- $L(M') = \Sigma^* \setminus L(M)$?



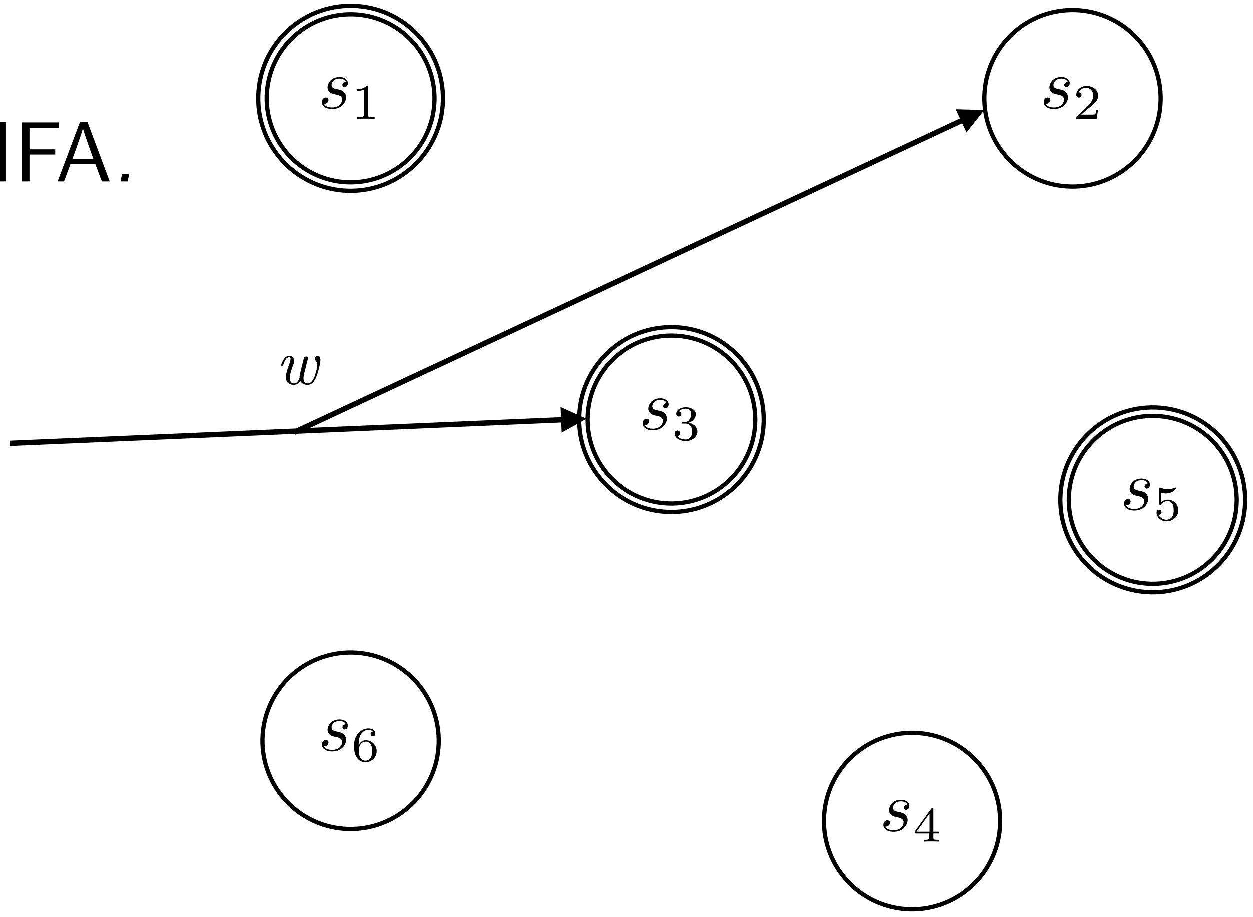
Complementation
NFA

- $M = (Q, \Sigma, \delta, I, F)$ is an NFA.
- $M' = (Q, \Sigma, \delta, I, Q \setminus F)$
- $L(M') = \Sigma^* \setminus L(M)$? ✗



Complementation
NFA

- $M = (Q, \Sigma, \delta, I, F)$ is an NFA.
- $M' = (Q, \Sigma, \delta, I, Q \setminus F)$
- $L(M') = \Sigma^* \setminus L(M)$? ✗



Exercise

- Let $M_1 = (Q_1, \Sigma, \delta_1, I_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, I_2, F_2)$ be two NFAs. Construct an NFA M_3 such that $L(M_3) = L(M_1) \setminus L(M_2)$. Please describe the components of M_3 in detail.

Minimization

- Given a DFA M_1, can we construct a minimal DFA M_2 such that $L(M_1) = L(M_2)$?

- Given an NFA M_1, can we construct a minimal NFA M_2 such that $L(M_1) = L(M_2)$?

Minimization

- Given a DFA M_1, can we construct a minimal DFA M_2 such that $L(M_1) = L(M_2)$?

- Given an NFA M_1, can we construct a minimal NFA M_2 such that $L(M_1) = L(M_2)$?

Minimization

- Given a DFA M_1, can we construct a minimal DFA M_2 such that $L(M_1) = L(M_2)$? ◯

- Given an NFA M_1, can we construct a minimal NFA M_2 such that $L(M_1) = L(M_2)$? ◯ **but harder**

Myhill-Nerode Theorem

- Given a language $L \subseteq \Sigma^*$, define a binary relation R_L over Σ^* as follows.
  - xR_Ly *iff* $\forall z \in \Sigma^* \; (xz \in L \leftrightarrow yz \in L)$
- R_L can be shown to be an equivalence relation.
- R_L divide the set of string into *equivalence classes*.
- L is regular iff R_L has a finite number of equivalence classes.
- The number of states in the minimal DFA recognizing L is equal to the number of equivalence classes in R_L.

Minimization
Idea

- For a language $L \subseteq \Sigma^*$, compute the equivalence classes of L.
- Construct a state for each equivalence class.
- A equivalence class C_1 can take an a-transition to another equivalence class C_2 if there is a string $x \in C_1$ such that $xa \in C_2$.
- How to find the equivalence classes?

Minimization

Hopcroft's Algorithm

```
P := {F, Q \ F};
W := {F};
while (W is not empty) do
     choose and remove a set A from W
     for each c in Σ do
          let X be the set of states for which a transition on c leads to a state in A
          for each set Y in P for which X ∩ Y is nonempty and Y \ X is nonempty do
               replace Y in P by the two sets X ∩ Y and Y \ X
               if Y is in W
                    replace Y in W by the same two sets
               else
                    if |X ∩ Y| <= |Y \ X|
                         add X ∩ Y to W
                    else
                         add Y \ X to W
          end;
     end;
end;
```

the pseudocode is taken from https://en.wikipedia.org/wiki/DFA_minimization

Language Expressions

- So far we know that a regular language can be accepted by a finite state automaton.
- Can we represent a regular language in other forms?

Language Expressions

- So far we know that a regular language can be accepted by a finite state automaton.

- Can we represent a regular language in other forms?

regular expressions

Regular Expressions (RE)

- Let Σ be an alphabet.
- The regular expressions over Σ are defined as follows.
  - $\emptyset$ is a regular expression denoting the empty set;
  - ϵ is a regular expression denoting the set $\{\epsilon\}$;
  - for each $a \in \Sigma$, a is a regular expression denoting the set $\{a\}$;
  - if r and s are regular expressions denoting the sets R and S respectively, then $r+s$, rs, and r^* are regular expressions denoting $R \cup S$, RS, and R^* respectively.
- The language of a regular expression e is denoted by $L(e)$.

Regular Expressions Examples

- Let $\Sigma = \{a,\ b\}$.
- $a^*ba^* = \{w \mid w$ *has exactly a single* $b\}$
- $\Sigma^*b\Sigma^* = \{w \mid w$ *has at least one* $b\}$
- $\Sigma^*aba\Sigma^* = \{w \mid w$ *has a substring* $aba\}$
- $a{+}b{+}a\Sigma^*a{+}b\Sigma^*b = \{w \mid w$ *starts and ends with the same symbol*$\}$

Regular Expressions Examples (cont'd)

- $r + \emptyset = ?$
- $r + \epsilon = ?$
- $r\emptyset = ?$
- $r\epsilon = ?$

Regular Expressions Examples (cont'd)

- $r + \emptyset = ?$ $\quad r$
- $r + \epsilon = ?$
- $r\emptyset = ?$
- $r\epsilon = ?$

Regular Expressions Examples (cont'd)

- $r + \emptyset = ?$ $\quad r$
- $r + \epsilon = ?$ $\quad r + \epsilon$
- $r\emptyset = ?$
- $r\epsilon = ?$

Regular Expressions Examples (cont'd)

- $r+\varnothing = ?$ $\quad r$
- $r+\epsilon = ?$ $\quad r+\epsilon$
- $r\varnothing = ?$ $\quad \varnothing$
- $r\epsilon = ?$

Regular Expressions Examples (cont'd)

- $r+\emptyset = ?$ $\quad r$
- $r+\epsilon = ?$ $\quad r+\epsilon$
- $r\emptyset = ?$ $\quad \emptyset$
- $r\epsilon = ?$ $\quad r$

Exercise

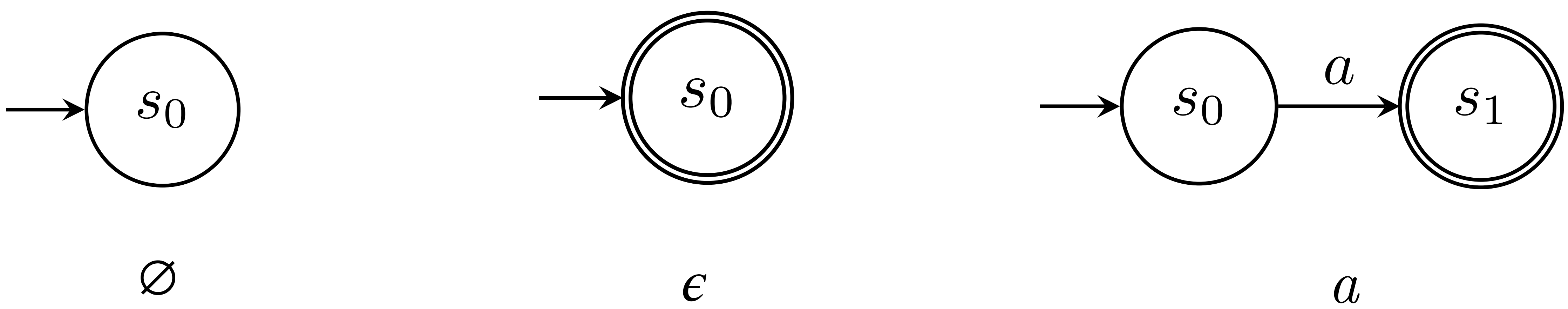
- Write regular expressions to describe the following languages. $(\Sigma = \{a,\ b\})$
  - $\{w \mid \text{the length of } w \text{ is even}\}$
  - $\{w \mid\ w \text{ has at most two } b\text{'s}\}$
  - $\{w \mid \text{every } a \text{ in } w \text{ is followed by } b\}$

Regular Expressions
VS
Finite State Automata

- A language is recognized by an NFA if and only if some regular expression describes it.

- A language is regular if and only if some regular expression describes it.

From RE to NFA



Let A_r be an NFA recognizing the language of a regular expression r.

$r+s$: union of A_r and A_s

rs: concatenation of A_r and A_s

r*: the Kleene closure of A_r

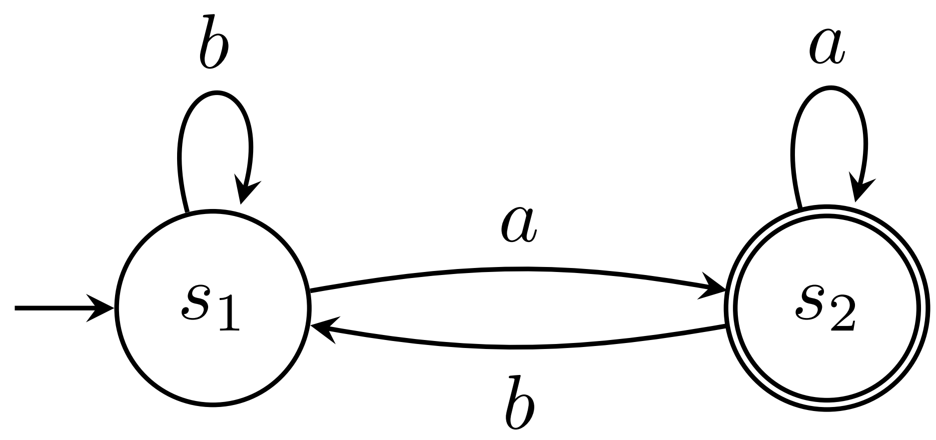
From NFA to RE

- Transitive Closure Method
- State Removal Method
- Brzozowski Algebraic Method

Transitive Closure Method

- Let $D = (\{s_1, \ldots, s_n\}, \Sigma, \delta, \{s_1\}, F)$ be a DFA.
- Define
  - $R_{ij}^{0} = \{a \mid (s_i, a, s_j) \in \delta\}$ if $i \neq j$
  - $R_{ij}^{0} = \{a \mid (s_i, a, s_j) \in \delta\} \cup \{\epsilon\}$ if $i = j$
  - $R_{ij}^{k} = R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$
- R_{ij}^{k} represents the inputs that cause D to go from s_i to s_j without passing through a state higher than s_k.
- R_{ij}^{k} can be denoted by regular expressions.
- $L(D) = \cup_{S_j \in F} R_{1j}^{n}$.

Transitive Closure Method Example



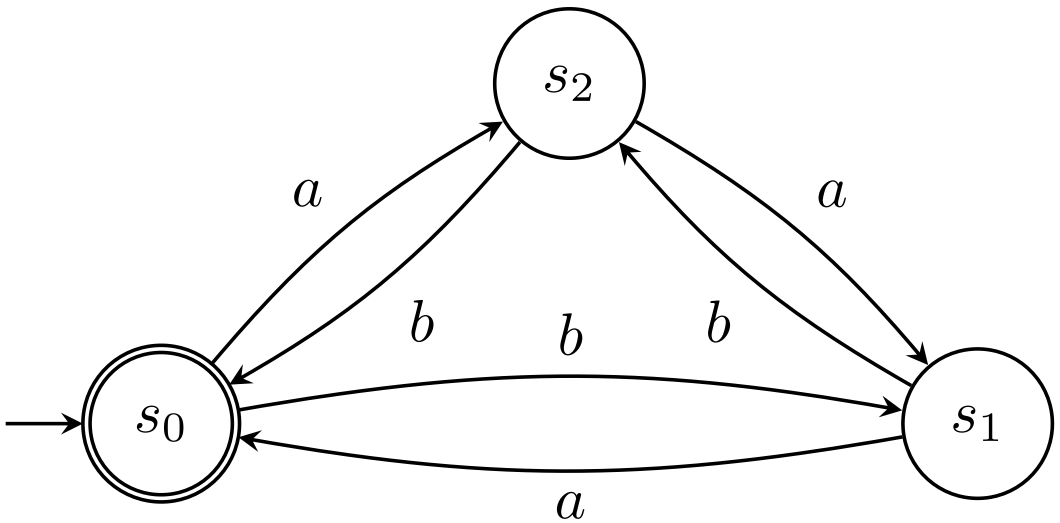
	$k=0$	$k=1$	$k=2$
${R_{11}}^k$	$b+\epsilon$	$(b+\epsilon)(b+\epsilon)^*(b+\epsilon)+(b+\epsilon)$ $= b^*$	
${R_{12}}^k$	a	$(b+\epsilon)(b+\epsilon)^*a+a$ $= b^*a$	$b^*a(b^*a+\epsilon)^*(b^*a+\epsilon)+b^*a$ $= (a+b)^*a$
${R_{21}}^k$	b	$b(b+\epsilon)^*(b+\epsilon)+b$ $= b^+$	
${R_{22}}^k$	$a+\epsilon$	$b(b+\epsilon)^*a+(a+\epsilon)$ $= b^*a+\epsilon$	

$b^+ = bb^*$

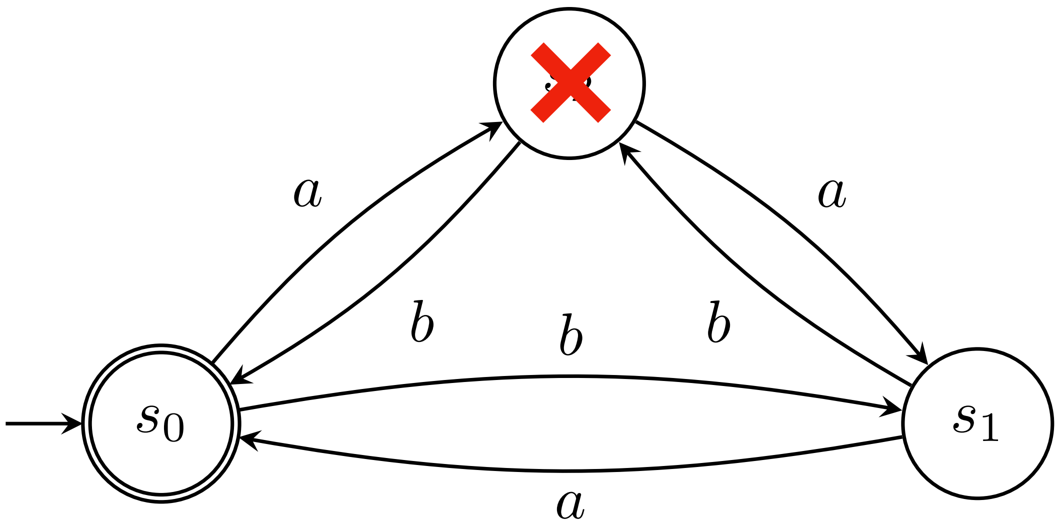
State Removal Method

- Make the NFA has a single accepting state.
- Make the NFA has a single initial state.
- Remove states and change transition labels (may be regular expressions) until there is only the initial state and the accepting state.
- Compute the regular expression.

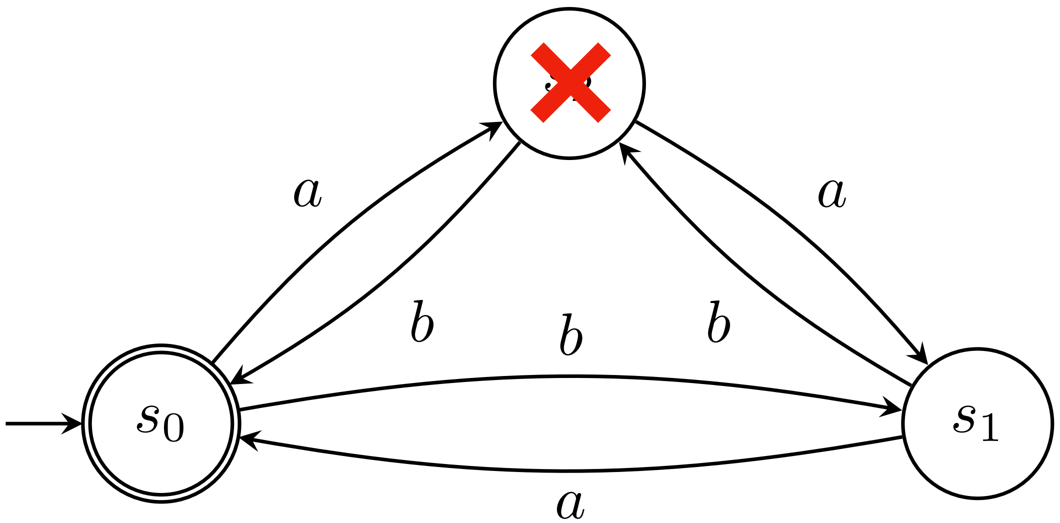
State Removal Method Example

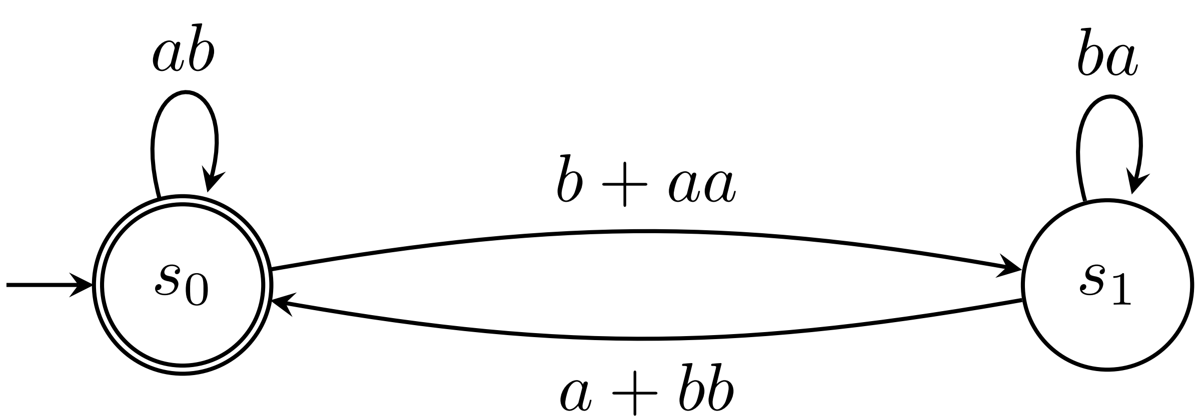


State Removal Method Example

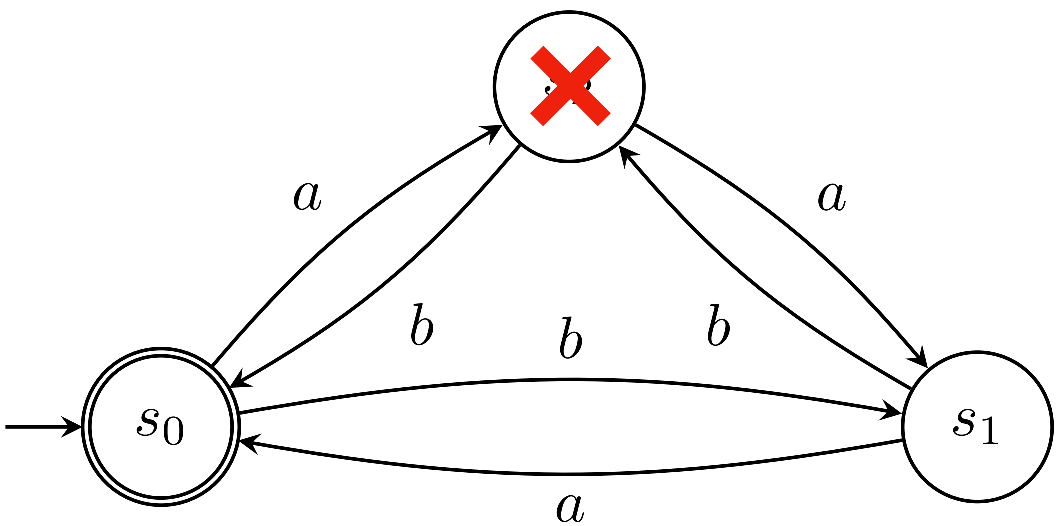


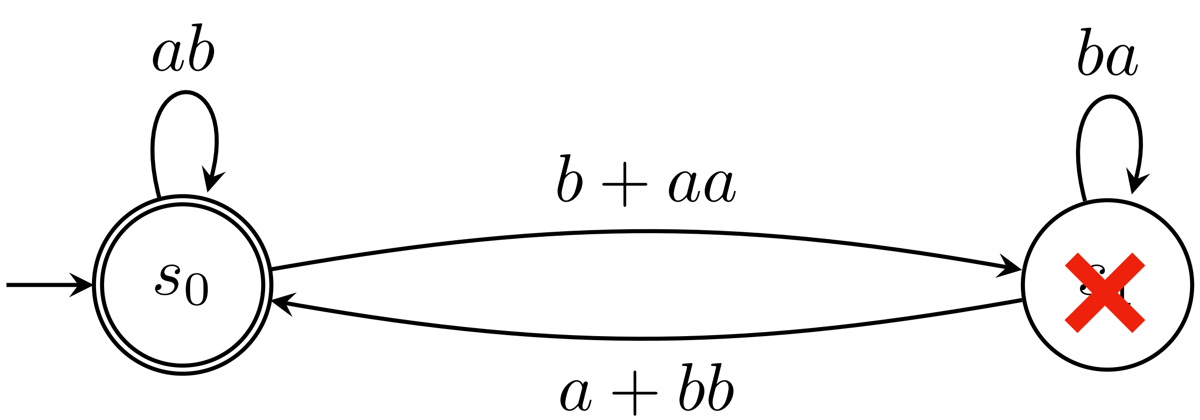
State Removal Method Example



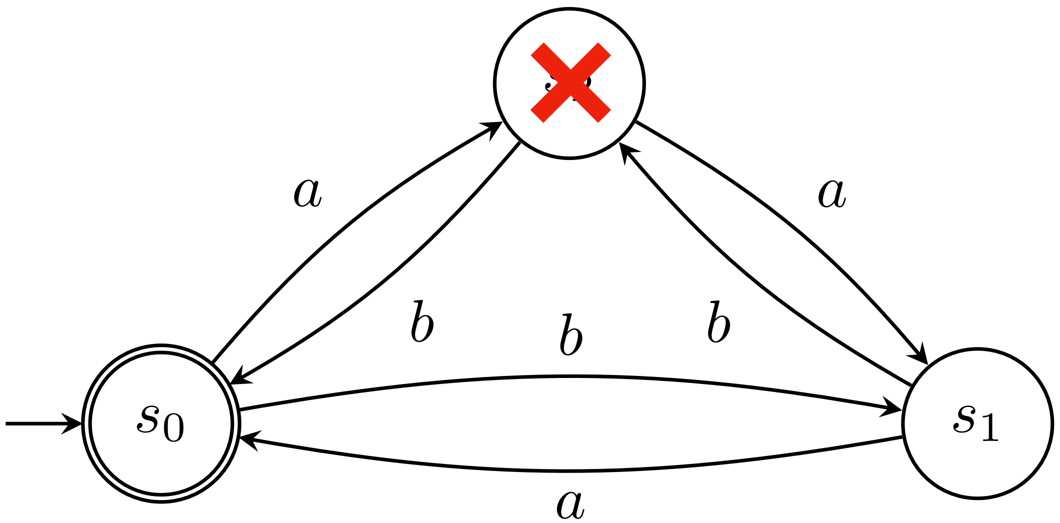


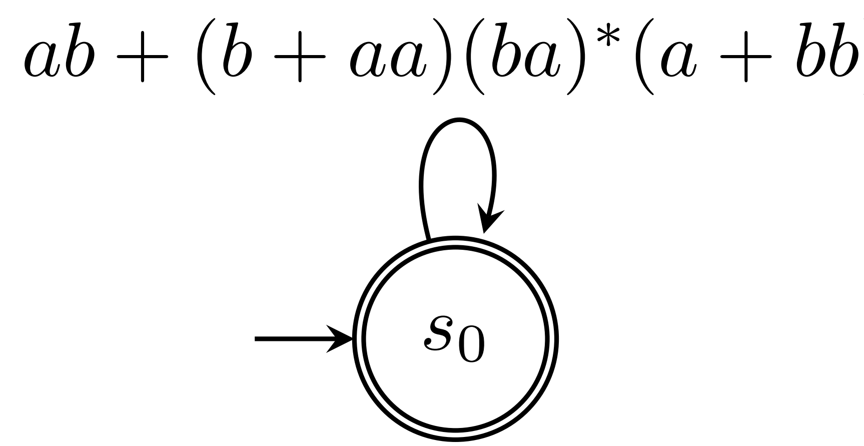
State Removal Method Example

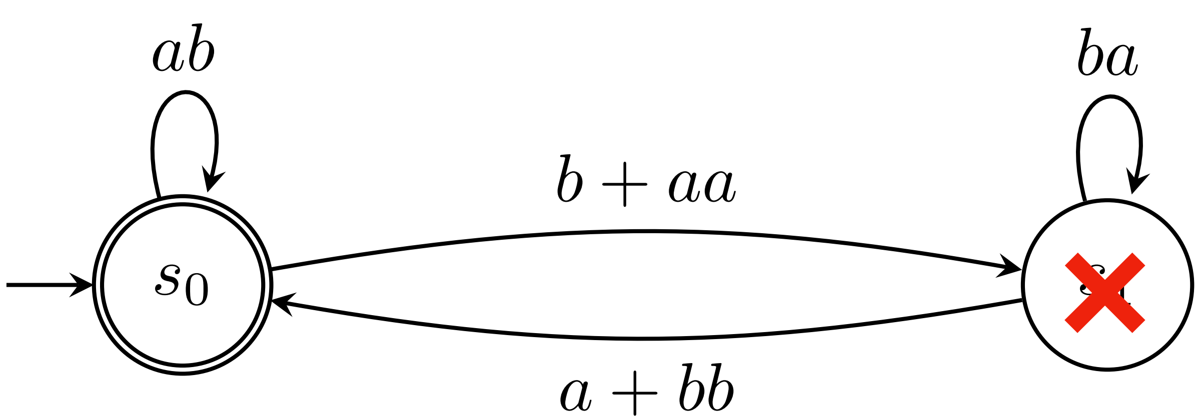




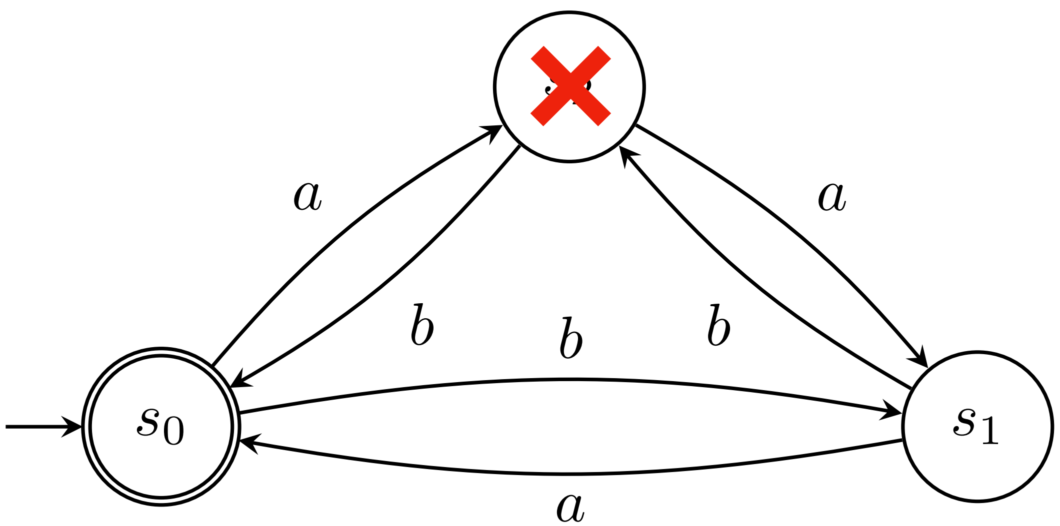
State Removal Method Example

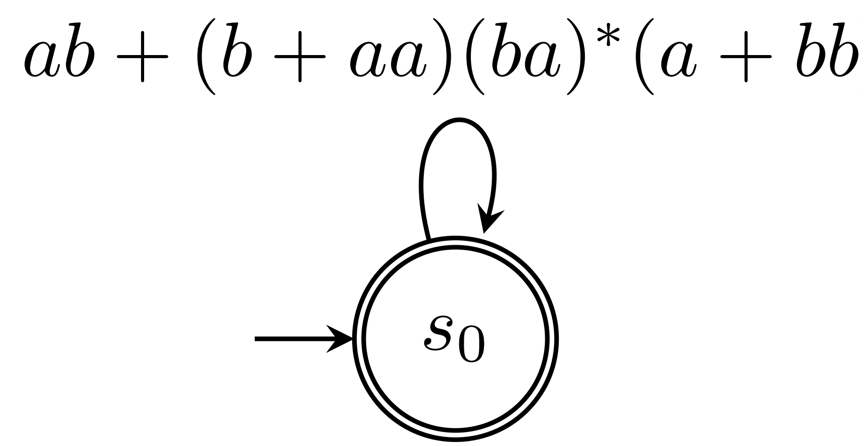


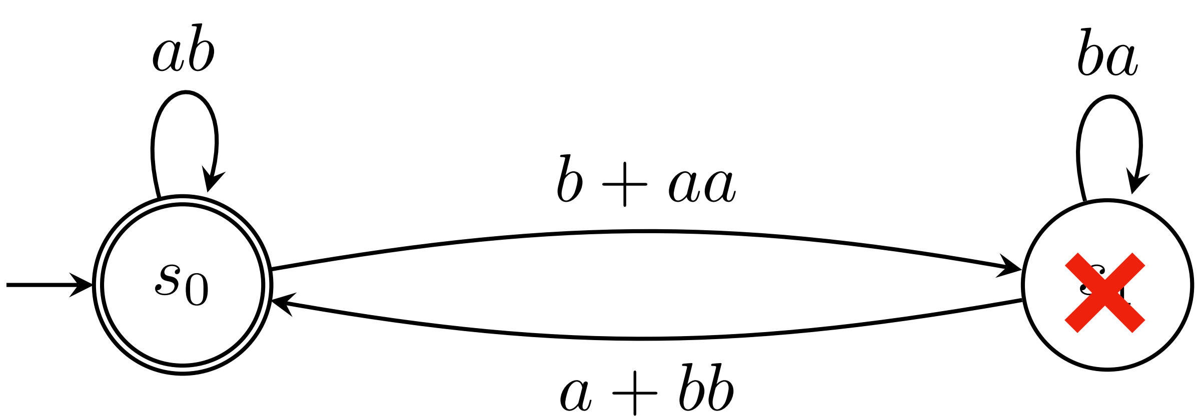




State Removal Method Example







$$(ab+(b+aa)(ba)^*(a+bb))^*$$

Brzozowski Algebraic Method

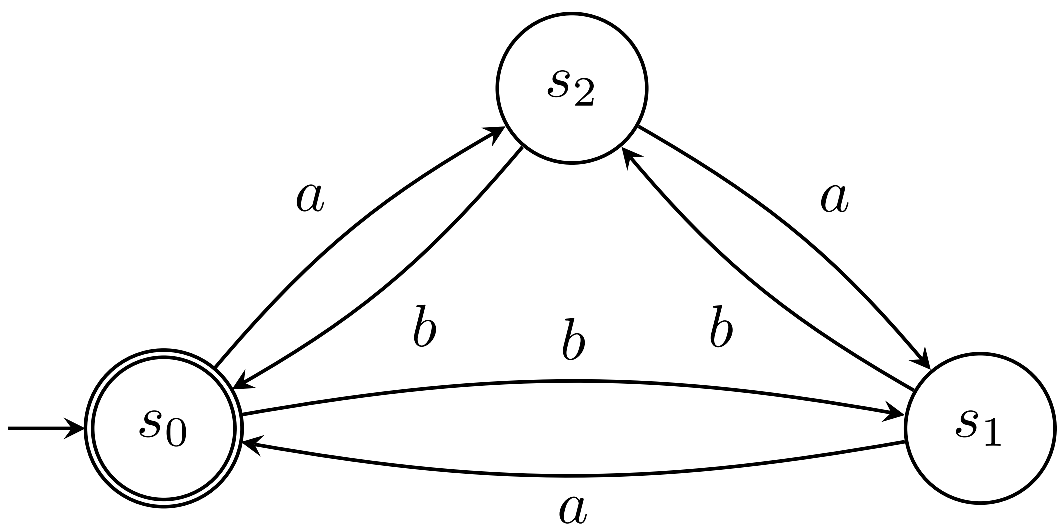
- $M = (Q, \Sigma, \delta, \{q_0\}, F)$ is an NFA containing no ϵ-transitions.

- For every q_i, create the equation

$$Q_i = +_{q_i \xrightarrow{a} q_j} aQ_j + \begin{cases} \{\epsilon\}, \text{if } q_i \in F \\ \varnothing, \text{else} \end{cases}$$

- Solve the equation system and find Q_0.

Brzozowski Algebraic Method Example



$$Q_0 = bQ_1 + aQ_2 + \epsilon$$
$$Q_1 = aQ_0 + bQ_2$$
$$Q_2 = bQ_0 + aQ_1$$

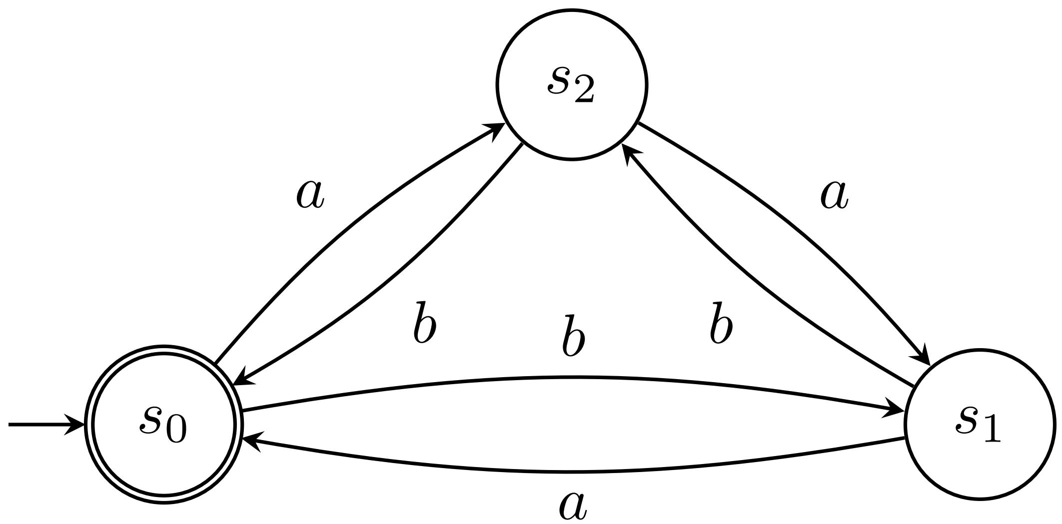
$$\begin{aligned} Q_2 &= bQ_0 + aQ_1 \\ &= bQ_0 + a(aQ_0 + bQ_2) \\ &= abQ_2 + (b+aa)Q_0 \end{aligned}$$

by Arden's Lemma:

$L = UL + V$ iff $L = U^* V$ where $L, U, V \subseteq \Sigma^*$ with $\epsilon \notin U$

$$Q_2 = (ab)^*(b+aa)Q_0$$

Brzozowski Algebraic Method Example (cont'd)



$$Q_0 = bQ_1 + aQ_2 + \epsilon$$
$$Q_1 = aQ_0 + bQ_2$$
$$Q_2 = bQ_0 + aQ_1$$

$$Q_2 = (ab)^*(b+aa)Q_0$$

$$\begin{aligned} Q_0 &= bQ_1 + aQ_2 + \epsilon \\ &= b(aQ_0 + bQ_2) + aQ_2 + \epsilon \\ &= baQ_0 + (bb+a)Q_2 + \epsilon \\ &= (ba+(bb+a)(ab)^*(b+aa))Q_0 + \epsilon \end{aligned}$$

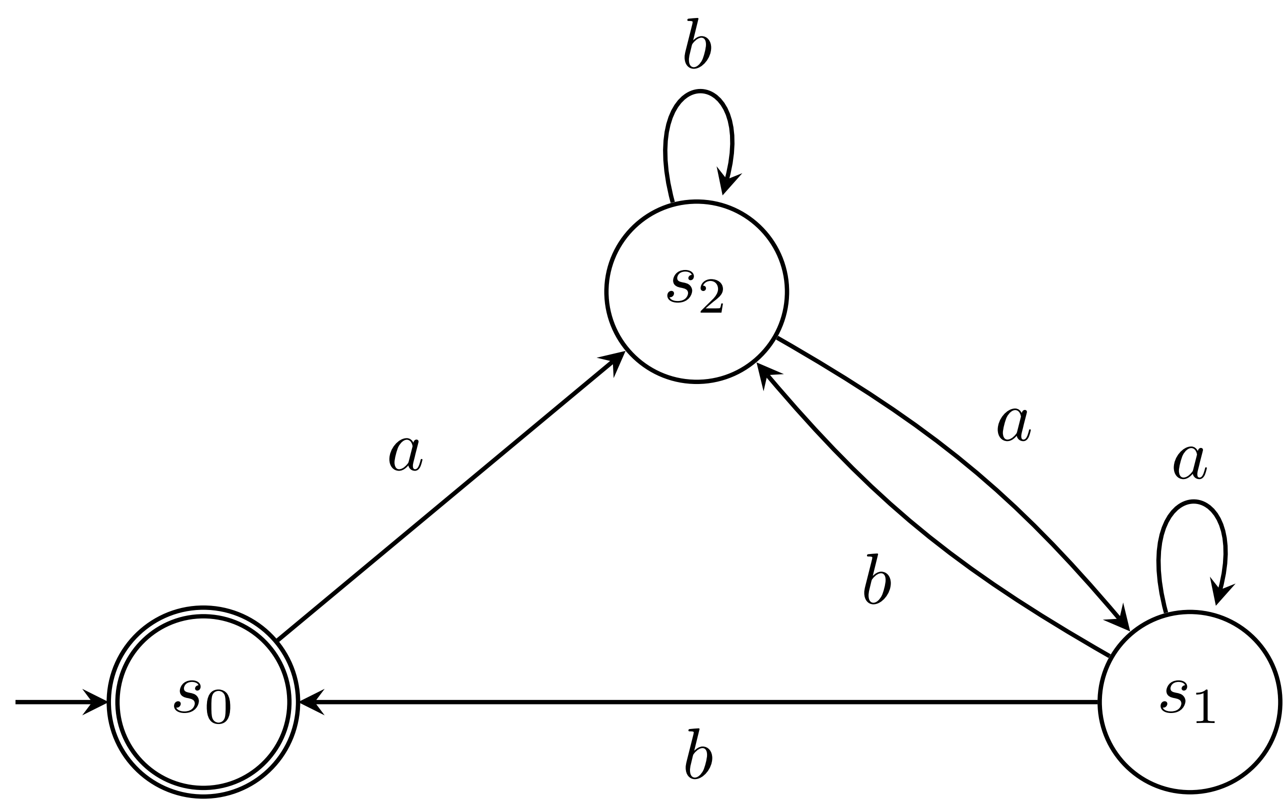
by Arden's Lemma:

$L = UL+V$ iff $L = U^*V$ where $L, U, V \subseteq \Sigma^*$ with $\epsilon \notin U$

$$Q_0 = (ba+(bb+a)(ab)^*(b+aa))^*$$

Exercise

- Express the language of the following automaton by a regular expression.



WS1S

- Syntax of S1S (monadic second-order logic of one successor)
  - First-order variable set: $V = \{x_1, x_2, ...\}$
  - Second-order variable set: $X = \{X_1, X_2, ...\}$
  - Terms: $t ::= 0 \mid x_i$
  - Formulas: $\varphi ::= S(t, t) \mid X_i(t) \mid \neg \varphi \mid \varphi \wedge \varphi \mid \exists x_i.\varphi \mid \exists X_i.\varphi$
- S is the successor predicate.
- WS1S: fragment of S1S which allows only quantification over finite sets

Semantics of S1S

- Signature $\langle \mathbb{N}, S \rangle$

- Interpretation $\sigma = \langle \sigma_1, \sigma_2 \rangle, \sigma_1 : V \to \mathbb{N}, \sigma_2 : X \to 2^{\mathbb{N}}$

- *Satisfiability*

$$
\begin{array}{lll}
\sigma \models X(t) & \textit{iff} & \sigma(t) \in \sigma(X) \\
\sigma \models S(t, t') & \textit{iff} & \sigma(t) + 1 = \sigma(t') \\
\sigma \models \neg\varphi & \textit{iff} & \sigma \not\models \varphi \\
\sigma \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma \models \exists x.\varphi & \textit{iff} & \sigma[n/x] \models \varphi \text{ for some } n \in \mathbb{N} \\
\sigma \models \exists X.\varphi & \textit{iff} & \sigma[N/X] \models \varphi \text{ for some } N \in 2^{\mathbb{N}}
\end{array}
$$

- *Validity* $\models \varphi \quad \textit{iff} \quad \sigma \models \varphi$ for all interpretations σ

Abbreviations

$$
\begin{array}{lcl}
\varphi_1 \vee \varphi_2 & := & \neg(\neg\varphi_1 \wedge \neg\varphi_2) \\
\varphi_1 \rightarrow \varphi_2 & := & \neg\varphi_1 \vee \varphi_2 \\
\forall x.\varphi & := & \neg\exists x.\neg\varphi \\
\forall X.\varphi & := & \neg\exists X.\neg\varphi \\
x \leq y & := & \forall X.(y \in X \wedge \forall z.\forall z'.(z \in X \wedge S(z', z) \rightarrow z' \in X) \rightarrow X(x)) \\
x < y & := & x \leq y \wedge \neg(y \leq x) \\
\mathit{first}(x) & := & \neg\exists y.S(y, x) \\
\mathit{last}(x) & := & \neg\exists y.S(x, y) \\
X \subseteq Y & := & \forall x.(x \in X \rightarrow x \in Y) \\
X = Y & := & X \subseteq Y \wedge Y \subseteq X \\
X = \varnothing & := & \forall Z, X \subseteq Z \\
\mathit{sing}(X) & := & X \neq \varnothing \wedge \forall Y.(Y \subseteq X \rightarrow (X \subseteq Y \vee Y = \varnothing))
\end{array}
$$

WS1S on Words

- Let Σ be a finite set of alphabet.
- A word is defined as $w = a_0 a_1 \ldots a_{n\text{-}1}$.
- A unary predicate P_a is defined for every $a \in \Sigma$ such that $P_a(i)$ if and only if $a_i = a$.
- Domain of w: $dom(w) = \{0, \ldots, |w| - 1\}$
- Word model of w: $\langle dom(w), S^w, (P_a)_{a \in \Sigma} \rangle$
- Büchi Theorem: a language $L \subseteq \Sigma^*$ is regular if and only if L is expressible in WS1S.

WS1S Examples

- the last symbol is a
  - $\exists x.(P_a(x) \wedge \neg\exists y.(x < y))$
- contains substring ab
  - $\exists x.\exists y.(P_a(x) \wedge P_b(y) \wedge S(x,y))$
- has substring ba^*b
  - $\exists x.\exists y.(x < y \wedge P_b(x) \wedge P_b(y) \wedge \forall z((x < z \wedge z < y) \rightarrow P_a(z)))$
- non-empty word with a even length
  - $\exists f.\exists l.\exists X.(\mathit{first}(f) \wedge \mathit{last}(l) \wedge X(f) \wedge \neg X(l) \wedge \forall y.\forall z.(S(y,z) \rightarrow (X(y) \leftrightarrow \neg X(z))))$

Exercises

- Write WS1S formulas to describe the following words.
  - Only a's can occur between any two occurrences of b's
  - Has an odd length (please start with $\exists$)

From NFA to WS1S

- Let $M = (Q,\ \Sigma,\ \delta,\ \{s_0\},\ F)$ be an NFA.
- Assume $Q = \{s_0,\ s_1,\ ...,\ s_n\}$.
- Non-empty accepting words will satisfy the following formula.

$$
\begin{array}{llll}
\exists X_0 \ldots X_n. & (& \wedge_{i \neq j} & \forall .x \neg (x \in X_i \wedge x \in X_j) \\
 & & \wedge & \forall x.(\mathit{first}(x) \rightarrow x \in X_0) \\
 & & \wedge & \forall x.\forall y.(S(x,y) \rightarrow \vee_{(s_i, a, s_j) \in \delta}(x \in X_i \wedge x \in P_a \wedge y \in X_j)) \\
 & & \wedge & \forall x.(\mathit{last}(x) \rightarrow \vee_{(s_i, a, s_f) \in \delta; s_f \in F}(x \in X_i \wedge x \in P_a)))
\end{array}
$$

A Better Encoding

- Assume $|\Sigma| = 2^m$.
- A symbol is binary encoded as $(t_0, t_1, ..., t_{m\text{-}1})$.
- A word is defined as $w = a_0 a_1 \ldots a_{n\text{-}1}$.
- A unary predicate P_i is defined for every $i \in \{0,\ldots,m\text{-}1\}$ such that $P_i(j)$ if and only if the i-th track of a_j is 1.
- Example:
  - $m = 2$, $\Sigma = \{a, b, c, d\}$, $a = (00)$, $b = (01)$, $c = (10)$, $d = (11)$
  - $P_0 = \{0, 3, 4\}$, $P_1 = \{1, 4\}$
  - $w = (10)(01)(00)(10)(11) = cbacd$

Non-regular Languages

- Examples of non-regular languages:
  - $\{\ a^n b^n \mid n \in \mathbb{N}\ \}$
  - $\{\ w\#w \mid w \in \{a,\ b\}^*\ \}$
- How to prove that a language is non-regular?

Pumping Lemma

- If L is a regular language, then there is a number $p \geq 1$ (the pumping length) such that, if s is any string in L and $|s| \geq p$, then s may be divided as $s = xyz$ satisfying
  - for each $i \geq 0$, $xy^iz \in L$,
  - $|y| > 0$, and
  - $|xy| \leq p$.

Pumping Lemma Example

- Let's show that $L = \{ a^n b^n \mid n \in \mathbb{N} \}$ is non-regular.
- Assume L is regular and let $w = a^p b^p$.
- By pumping lemma, there are x, y, and z such that $w = xyz$,
  - $xy^i z \in L$ for each $i \geq 0$,
  - $|y| \geq 0$, and
  - $|xy| \leq p$.
- With $|xy| \leq p$, we know that y contains only a.
- But $xy^2 z \notin L$.

Formal Languages

Chomsky Hierarchy	Grammar	Language	Computation Model
Type-0	Unrestricted	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

the list of formal languages in this table is not complete

Tools

- MONA (http://www.brics.dk/mona/)
- JFLAP (http://www.jflap.org)

Infinite Computations

- A *reactive system* is a system that continuously interacts with its environment.
- Computations of a reactive system are infinite.
- How to model such infinite computations?
  - Automata on infinite words

Infinite Words

- Let Σ be a finite alphabet.
- An infinite word *w* over Σ ($w \in \Sigma^{\omega}$) is a sequence of symbols $a_0 a_1 a_2 ...$ with $a_i \in \Sigma$.
  - Length of w is ω.
- Examples ($\Sigma = \{a,\ b\}$):
  - $a\ b\ (b\ a)^{\omega}$
  - $a\ b\ a\ (b\ a\ b)^{\boldsymbol{\omega}}$

ω-Automata

Syntax

- An *ω-automaton* is a tuple $(Q, \Sigma, \delta, q_0, Acc)$ where
  - Q is a finite set of states,
  - Σ is a finite alphabet,
  - δ: $Q \times \Sigma \to 2^Q$ is the transition function,
  - q_0 is the initial state, and
  - Acc is the *acceptance condition*.
- Different ω-automata can be defined by different acceptance conditions.

ω-Automata Semantics

- Let $M = (Q,\ \Sigma,\ \delta,\ q_0,\ Acc)$ be an ω-automaton.
- Let $w = a_0 a_1 a_2 ...$ be an infinite word over Σ.
- A *run* of w on M is a sequence of states $q_0 q_1 q_2 ...$ where $(q_i,\ a_i,\ q_{i+1}) \in \delta$.

ω-Automata
Semantics (cont'd)

- A run is accepting if the run satisfies the acceptance condition Acc.

- A word is accepted if there is a run of M on the word.

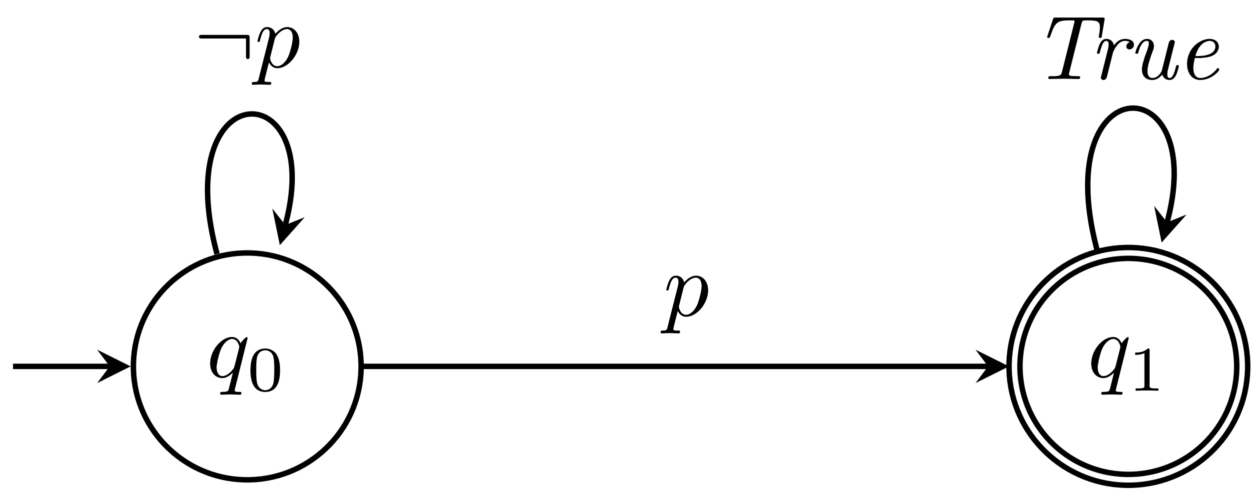
- The language of M, denoted by $L(M)$, is the set of words accepted by M.

- Define $\mathit{Inf}(\rho) = \{s \mid s \text{ *occurs in* } \rho \text{ *infinitely many times*}\}$.

Acceptance Conditions

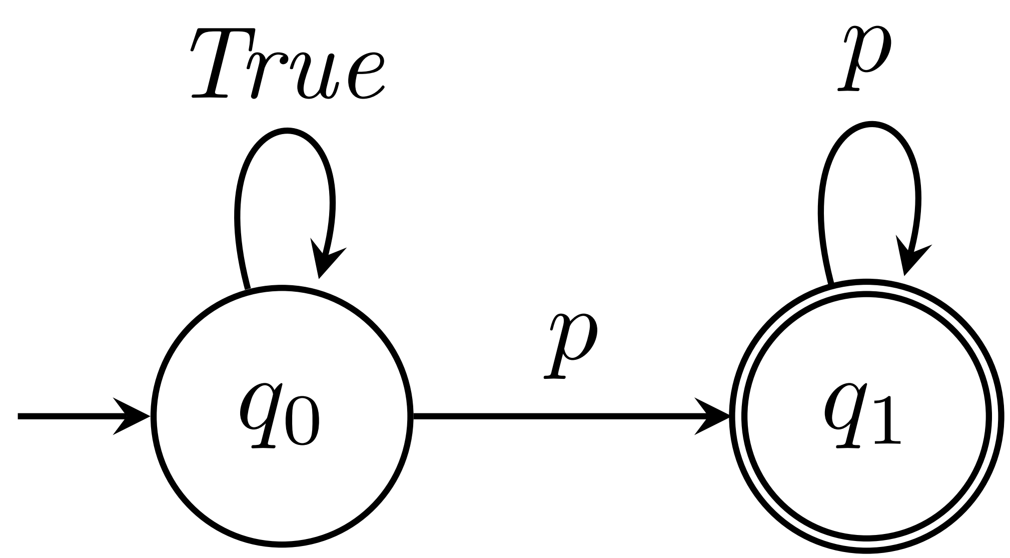
Acceptance Condition	Acc	Satisfaction	Abbrev.	Note
Büchi	$Acc = F \subseteq Q$	$Inf(\rho) \cap F \neq \varnothing$	NBW	
co-Büchi	$Acc = F \subseteq Q$	$Inf(\rho) \cap F = \varnothing$	NCW	
Generalized Büchi	$Acc = \{F_1, \ldots, F_n\}$, $F_i \subseteq Q$	$Inf(\rho) \cap F_i \neq \varnothing$ for all $F_i \in F$	NGW	
Rabin	$Acc = \{(E_1, F_1), \ldots, (E_n, F_n)\}$, $F_i \subseteq Q$, $E_i \subseteq Q$	$Inf(\rho) \cap E_i = \varnothing$ and $Inf(\rho) \cap F_i \neq \varnothing$ for some i	NRW	
Streett	$Acc = \{(E_1, F_1), \ldots, (E_n, F_n)\}$, $F_i \subseteq Q$, $E_i \subseteq Q$	$Inf(\rho) \cap F_i \neq \varnothing$ implies $Inf(\rho) \cap E_i \neq \varnothing$ for all i	NSW	
Muller	$Acc = \{F_1, \ldots, F_n\}$, $F_i \subseteq Q$	$Inf(\rho) = F_i$ for some i	NMW	
Parity	$Acc\colon Q \to \mathbb{N}$	min parity in ρ is even	NPW	$Acc(q)$ is the parity of q

Büchi Automata
Example 1



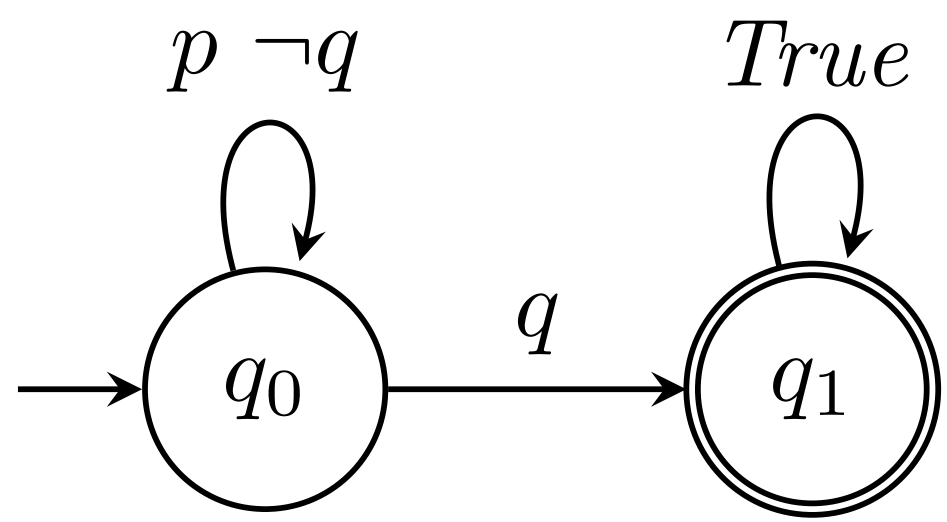
accepts infinite words where p holds eventually

Büchi Automata
Example 2



accepts infinite words where eventually p will always hold

Büchi Automata
Example 3



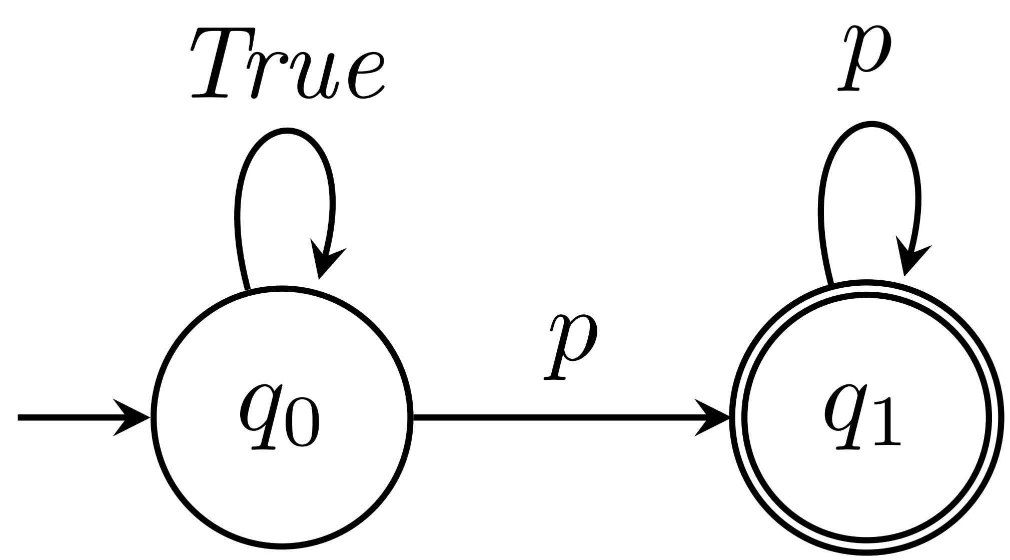
accepts infinite words where p holds until q holds

Exercise

- Draw a Büchi automaton that accepts infinite words where p holds infinitely many times. ($\Sigma = \{p, \neg p\}$)

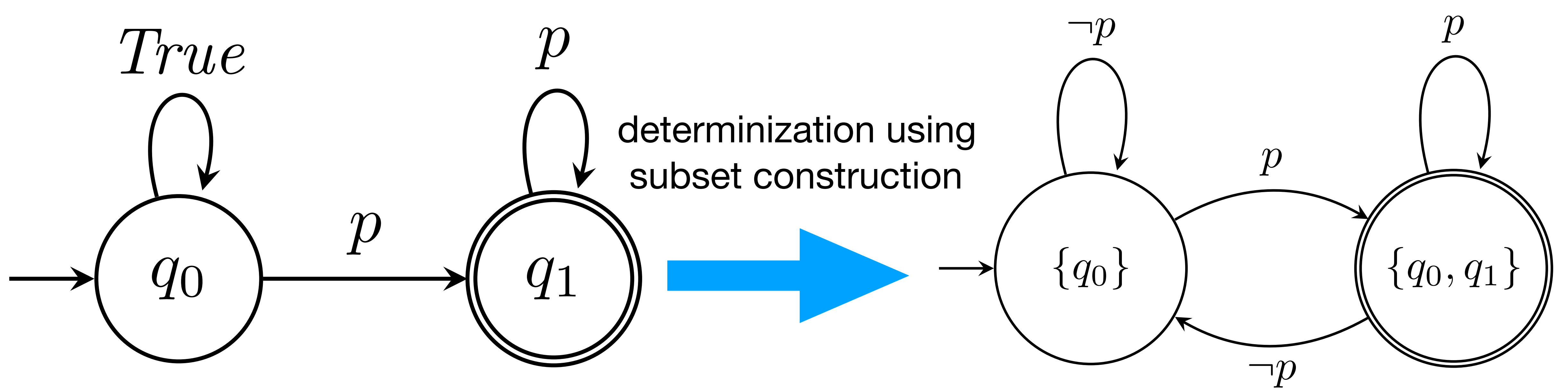
Deterministic VS Nondeterministic

- Can you find a deterministic Büchi automaton (DBW) that accepts the same language?



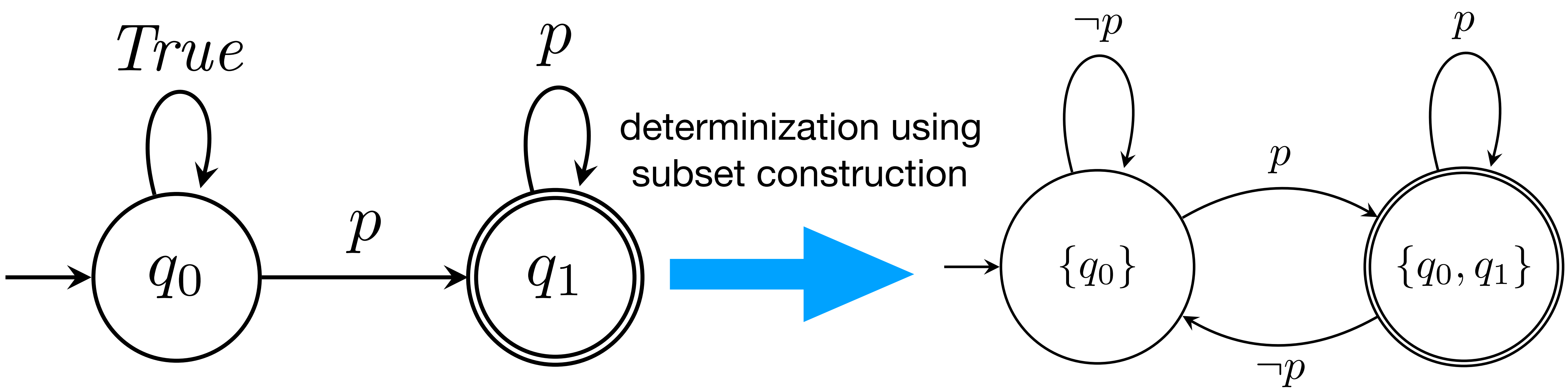
Deterministic VS Nondeterministic

- Can you find a deterministic Büchi automaton (DBW) that accepts the same language?



Deterministic VS Nondeterministic

- Can you find a deterministic Büchi automaton (DBW) that accepts the same language?



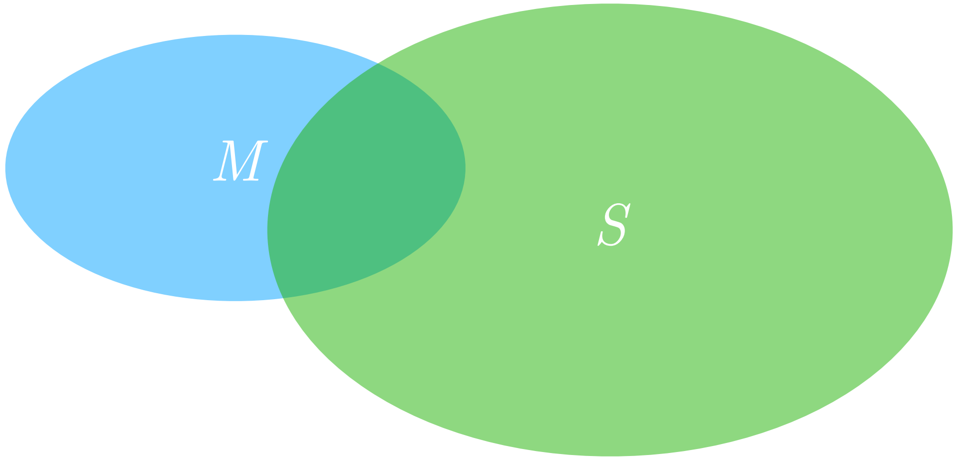
NBW is more expressive than DBW

Model VS Specification

- So far we already learnt some abstract machines as models of computations.
- We may require that the computations must satisfy some properties.
- How do we check?

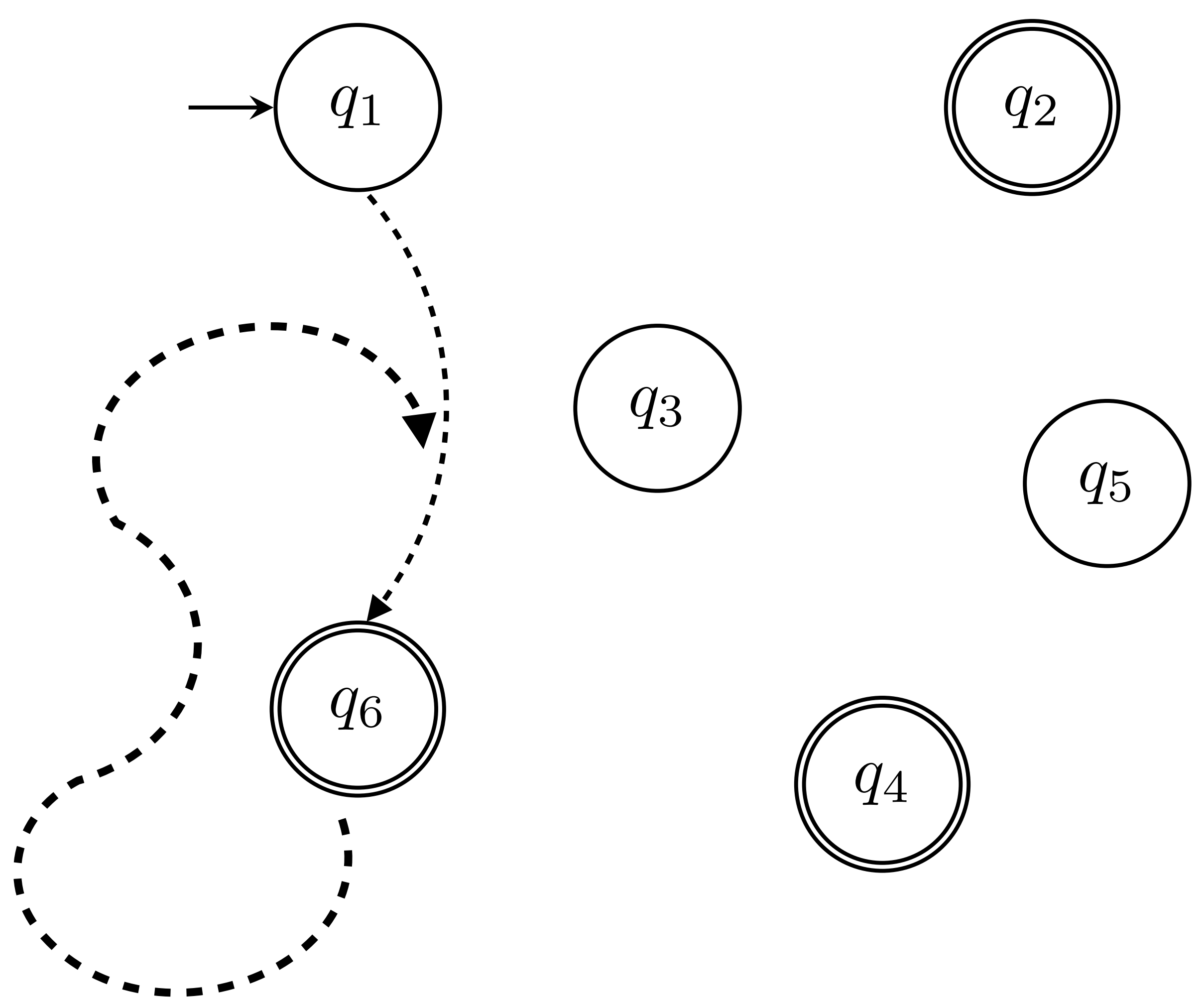
Model Checking

- Model the computations of a system as an automaton M.
- Model the computations allowed by the specification as an automaton S.
- Check if the system satisfies the specification by checking if $L(M) \subseteq L(S)$.
- Or equivalently checking if P is empty where P is the intersection of
  - M and
  - the complement of S.



Emptiness Test

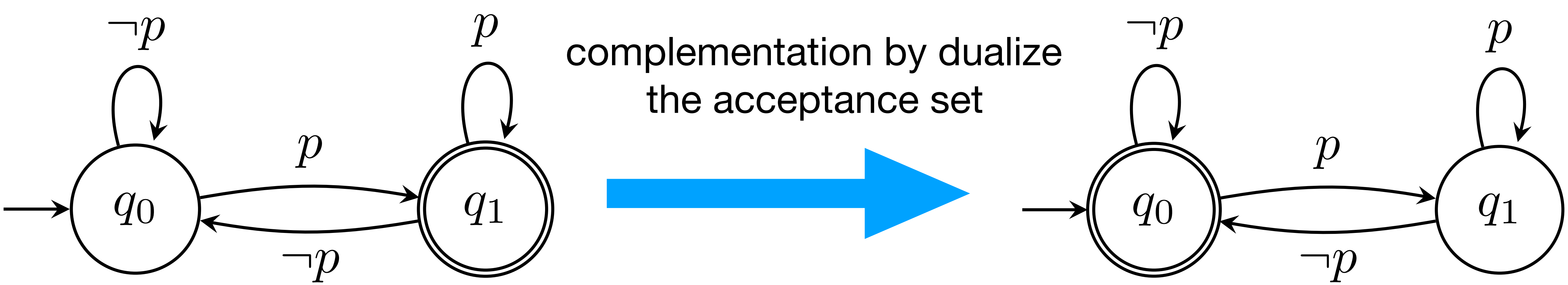
- Use double depth-first search to find an accepting lasso.



Büchi Automata Intersection

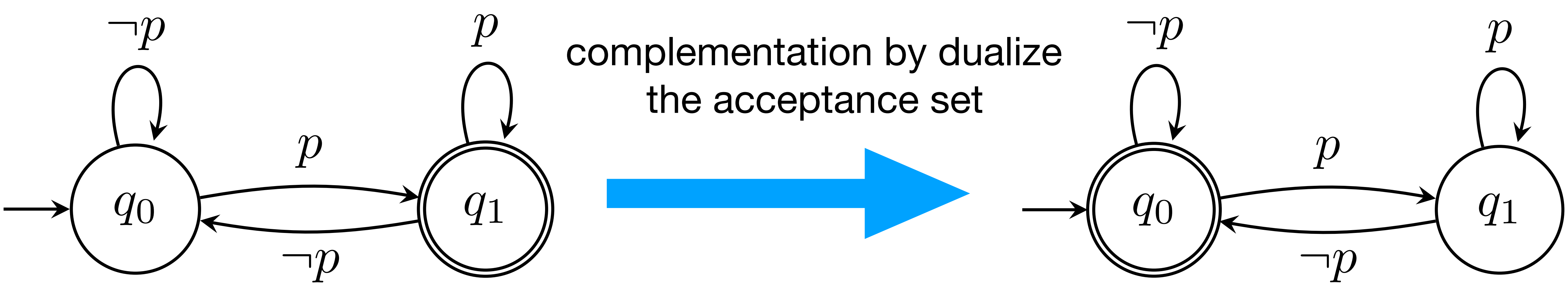
- $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$
- Construct $M = (Q_1 \times Q_2 \times \{0,1,2\}, \Sigma, \delta, (q_{01}, q_{02}, 0), Q_1 \times Q_2 \times \{0\})$ where $((q_1, q_2, i), a, (q_1', q_2', j)) \in \delta$ if
  - $(q_1, a, q_1') \in \delta_1$ and $(q_2, a, q_2') \in \delta_2$,
  - $j = 1$ if $i = 0$,
  - $j = i$ if $i \neq 0$ and $q_i \notin F_i$, and
  - $j = (i + 1)\ mod\ 2$ if $i \neq 0$ and $q_i \in F_i$.
- $L(M) = L(M_1) \cap L(M_2)$

Büchi Automata Complementation



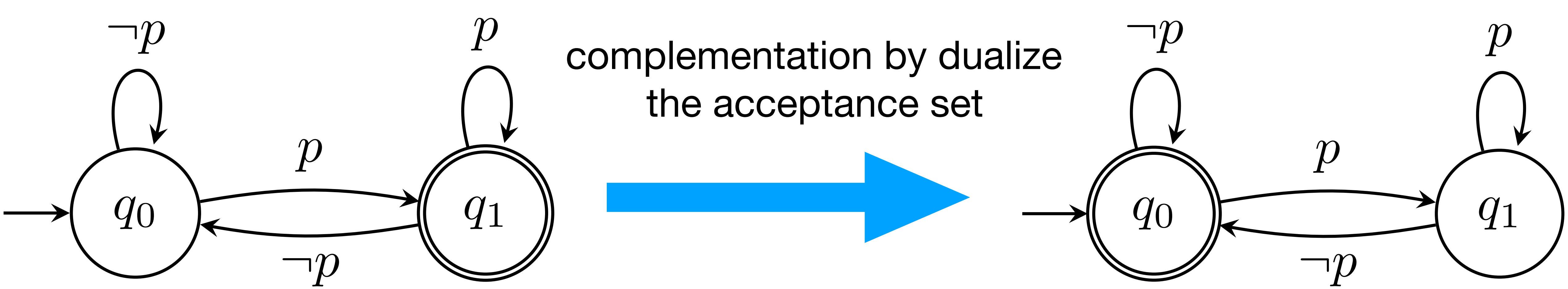
Does the right one exactly accept the complement of the left one?

Büchi Automata Complementation



Does the right one exactly accept the complement of the left one? ❌

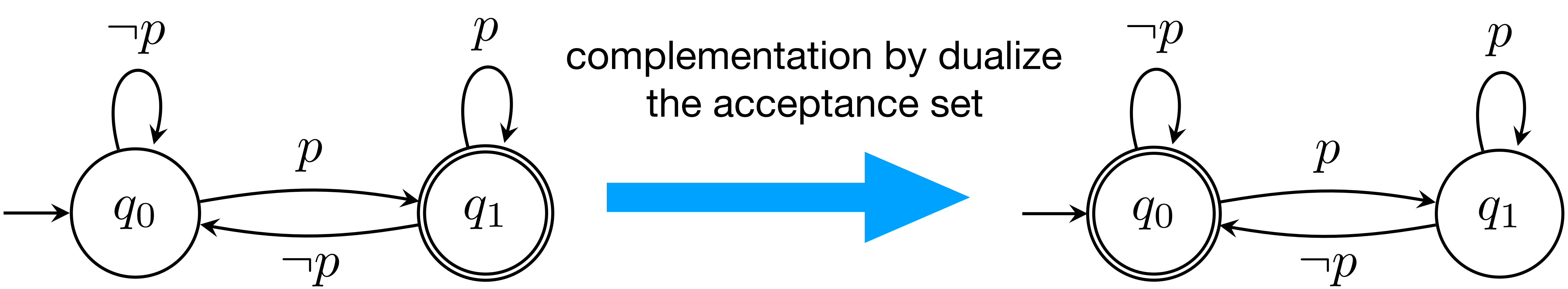
Büchi Automata Complementation



Does the right one exactly accept the complement of the left one? ✗

Complementation of NBW is much harder than that of NFA.

Büchi Automata Complementation



Does the right one exactly accept the complement of the left one? ❌

Complementation of NBW is much harder than that of NFA.

We may express specifications using logic formulas.

LTL Model Checking

- Express the behavior of a system as a Büchi automaton *M* (usually converted from a Kripke structure).
- Express the specification as a formula f in *linear temporal logic* (LTL).
- Translation $\neg f$ to a Büchi automaton $A_{\neg f}$ *with labels on states*.
- Check if $L(M) \cap L(A_{\neg f})$ is empty.

Linear Temporal Logic Syntax

- AP is a finite set of atomic propositions.
- The alphabet Σ is defined as 2^{AP}.
- A linear temporal logic (LTL) formula is defined as follows.
  - For every $p \in AP$, p is an LTL formula.
  - If f and g are LTL formulas, then so are $\neg f$, $f \wedge g$, $\boldsymbol{X} f$, and $f \boldsymbol{U} g$.
- $\boldsymbol{X}$ and $\boldsymbol{U}$ are (future) temporal operators.

Linear Temporal Logic Semantics

- A state is a subset of AP, containing exactly those propositions that evaluate to true in that state.

- An LTL formula is interpreted over an infinite sequence of states $\rho = s_0 s_1 \ldots$.

$(\rho,\ i) \models p$ iff $p \in s_i$

$(\rho,\ i) \models \neg\, f$ iff $(\rho,\ i) \not\models f$

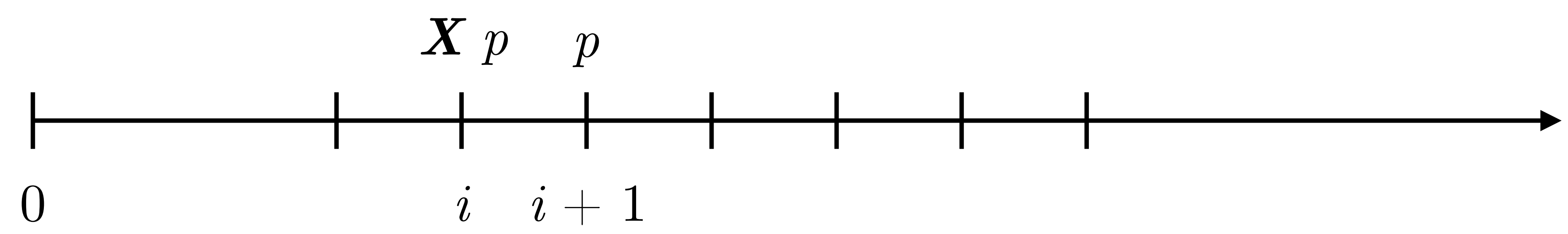
$(\rho,\ i) \models f \wedge g$ iff $(\rho,\ i) \models f$ and $(\rho,\ i) \models g$

$(\rho,\ i) \models \boldsymbol{X}\, f$ iff $(\rho,\ i+1) \models f$

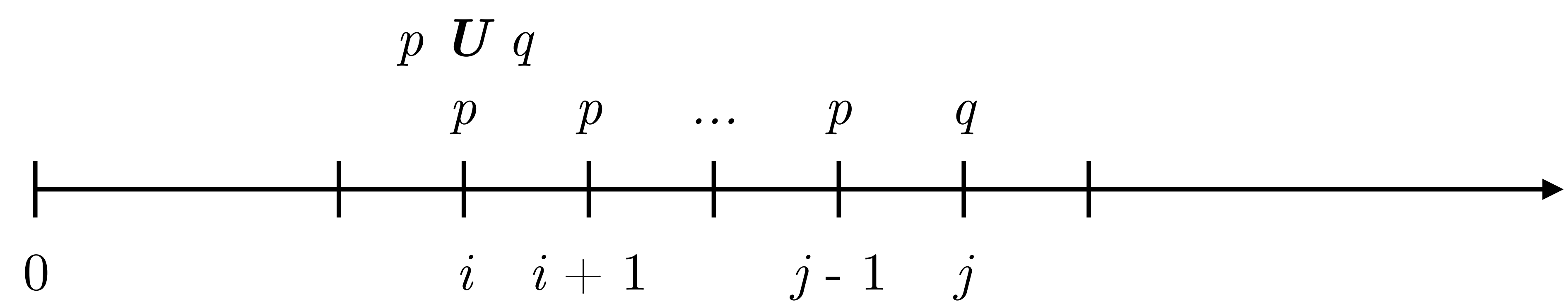
$(\rho,\ i) \models f\ \boldsymbol{U}\ g$ iff exists $j \geq i$ such that $(\rho,\ j) \models g$ and for all $i \leq k < j$, $(\rho,\ k) \models f$

Next and Until

- $(\rho,\ i) \vDash \boldsymbol{X}\, f$ iff $(\rho,\ i+1) \vDash f$

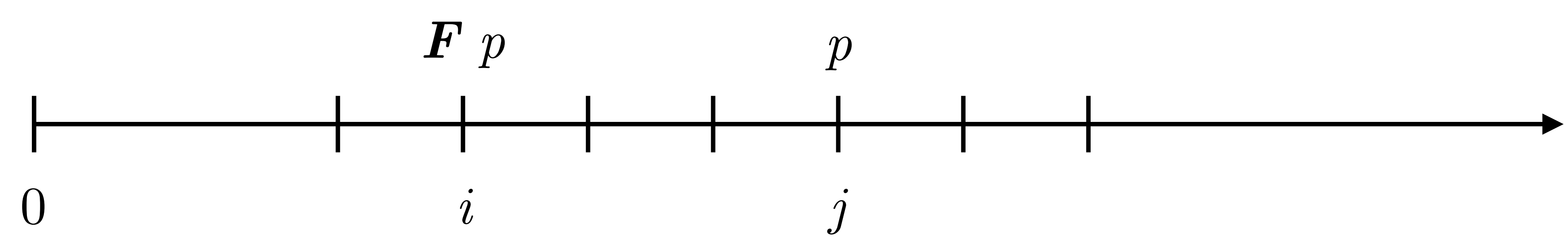


- $(\rho,\ i) \vDash f\ \boldsymbol{U}\ g$ iff exists $j \geq i$ such that $(\rho,\ j) \vDash g$ and for all $i \leq k < j$, $(\rho,\ k) \vDash f$

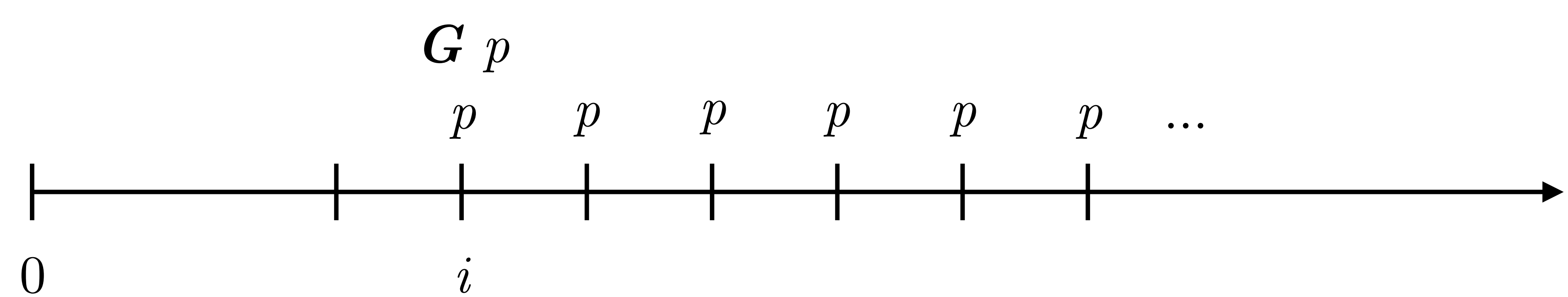


Future and Global

- $(\rho,\ i) \vDash \boldsymbol{F}\ f$ iff $(\rho,\ j) \vDash f$ for some $j \geq i$

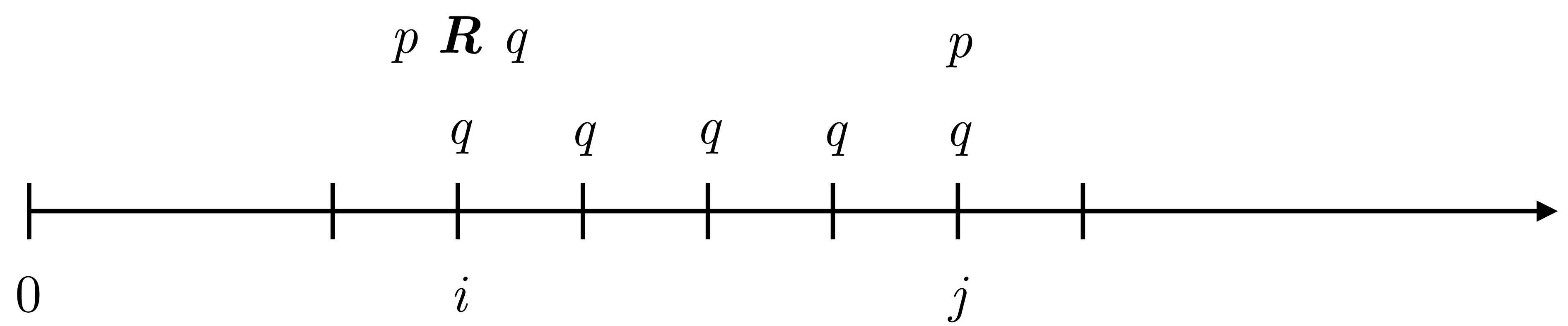


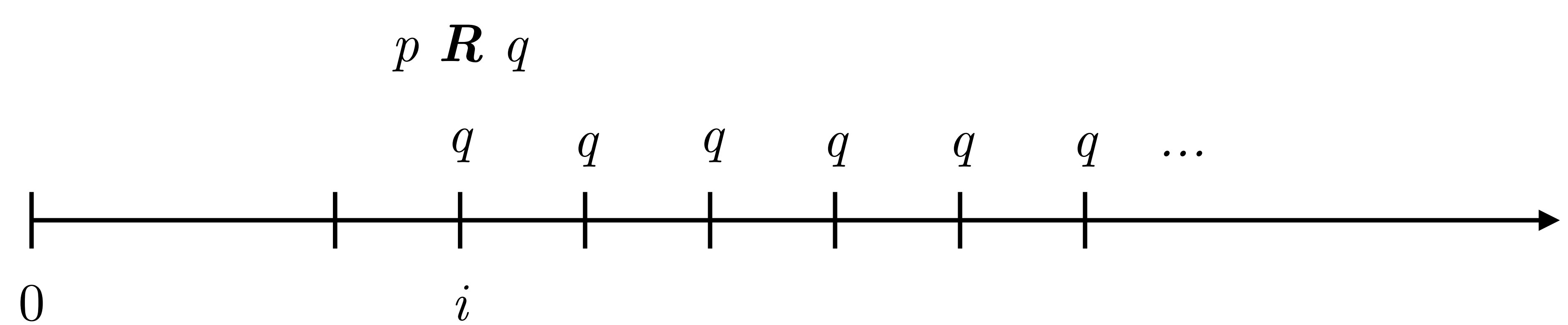
- $(\rho,\ i) \vDash \boldsymbol{G}\ f$ iff $(\boldsymbol{\rho},\ j) \vDash f$ for all $j \geq i$



Release

- $(\rho,\ i) \vDash f\ \boldsymbol{R}\ g$ iff exists $j \geq\ i$ such that $(\rho,\ j) \vDash f$ and for all $i \leq\ k \leq\ j$, $(\rho,\ k) \vDash g$; or for all $j \geq\ i$, $(\rho,\ j) \vDash g$





Abbreviations

- $true := p \vee \neg p$
- $false := \neg true$
- $f \vee g := \neg(\neg f \wedge \neg g)$
- $f \rightarrow g := \neg f \vee g$
- $f \leftrightarrow g := (f \rightarrow g) \wedge (g \rightarrow f)$
- $f \boldsymbol{R} g := \neg(\neg f \boldsymbol{U} \neg g)$
- $\boldsymbol{F} g := true \boldsymbol{U} g$
- $\boldsymbol{G} f := false \boldsymbol{R} f$

$\bigcirc = \boldsymbol{X}$, $\Diamond = \boldsymbol{F}$, $\Box = \boldsymbol{G}$

Exercise

- Express the following sentences in LTL formulas.
  - "p occurs infinitely often"
  - "whenever a message is sent, eventually an acknowledgement will be received"

Satisfaction, Validity, and Congruence

- $\rho \models f$: a state sequence ρ *satisfies* an LTL formula f
  - $\rho \models f$ iff $(\rho, 0) \models f$
- $\models f$: an LTL formula f is *valid*
  - $\models f$ iff $\rho \models f$ for all ρ
- $f \cong g$: two formulas f and g are *congruent*
  - $f \cong g$ iff $\models \boldsymbol{G}\ (f \leftrightarrow g)$

Congruent Formulas

- $\neg\ \boldsymbol{X}\ f \cong \boldsymbol{X}\ \neg\ f$
- $\neg\ \boldsymbol{F}\ g \cong \boldsymbol{G}\ \neg\ g$
- $\neg\ \boldsymbol{G}\ f \cong \boldsymbol{F}\ \neg\ f$
- $\boldsymbol{G}\ \boldsymbol{G}\ f \cong \boldsymbol{G}\ f$
- $\boldsymbol{F}\ \boldsymbol{F}\ g \cong \boldsymbol{F}\ g$
- $\neg\ \neg\ f \cong f$

Basic Formulas

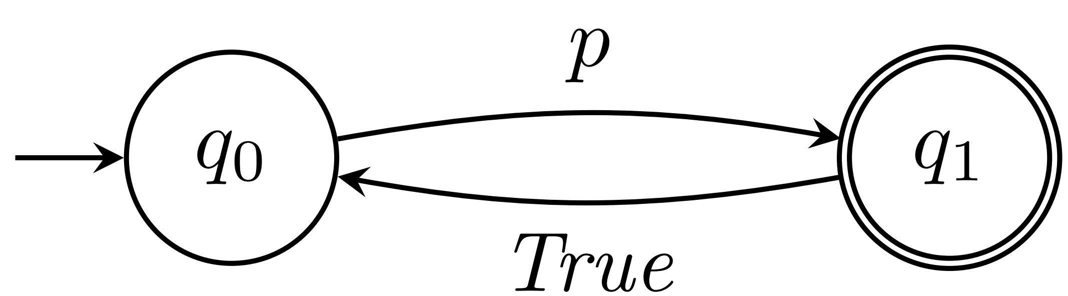
- A *literal* is either a proposition or its negation.
- A basic formula is either a literal or an $\boldsymbol{X}$-formula.

Expansion Formulas

- $\boldsymbol{F}\ g \cong g \vee \boldsymbol{X}\ \boldsymbol{F}\ g$
- $\boldsymbol{G}\ f \cong f \wedge \boldsymbol{X}\ \boldsymbol{G}\ f$
- $f\ \boldsymbol{U}\ g \cong g \vee (f \wedge \boldsymbol{X}\ (f\ \boldsymbol{U}\ g))$
- $f\ \boldsymbol{R}\ g \cong g \wedge (f \vee \boldsymbol{X}\ (f\ \boldsymbol{R}\ g))$

Expressive Power of LTL

- LTL is strictly less expressive than NBW.
- "even p" can be expressed in NBW but not LTL.



- NBW is as expressive as QPTL (Quantified Propositional Temporal Logic).
- "even p" in QPTL: $\exists\ t.\ t \wedge \boldsymbol{G}\ (t \leftrightarrow \boldsymbol{X}\ \neg t) \wedge \boldsymbol{G}\ (t \rightarrow\ p)$

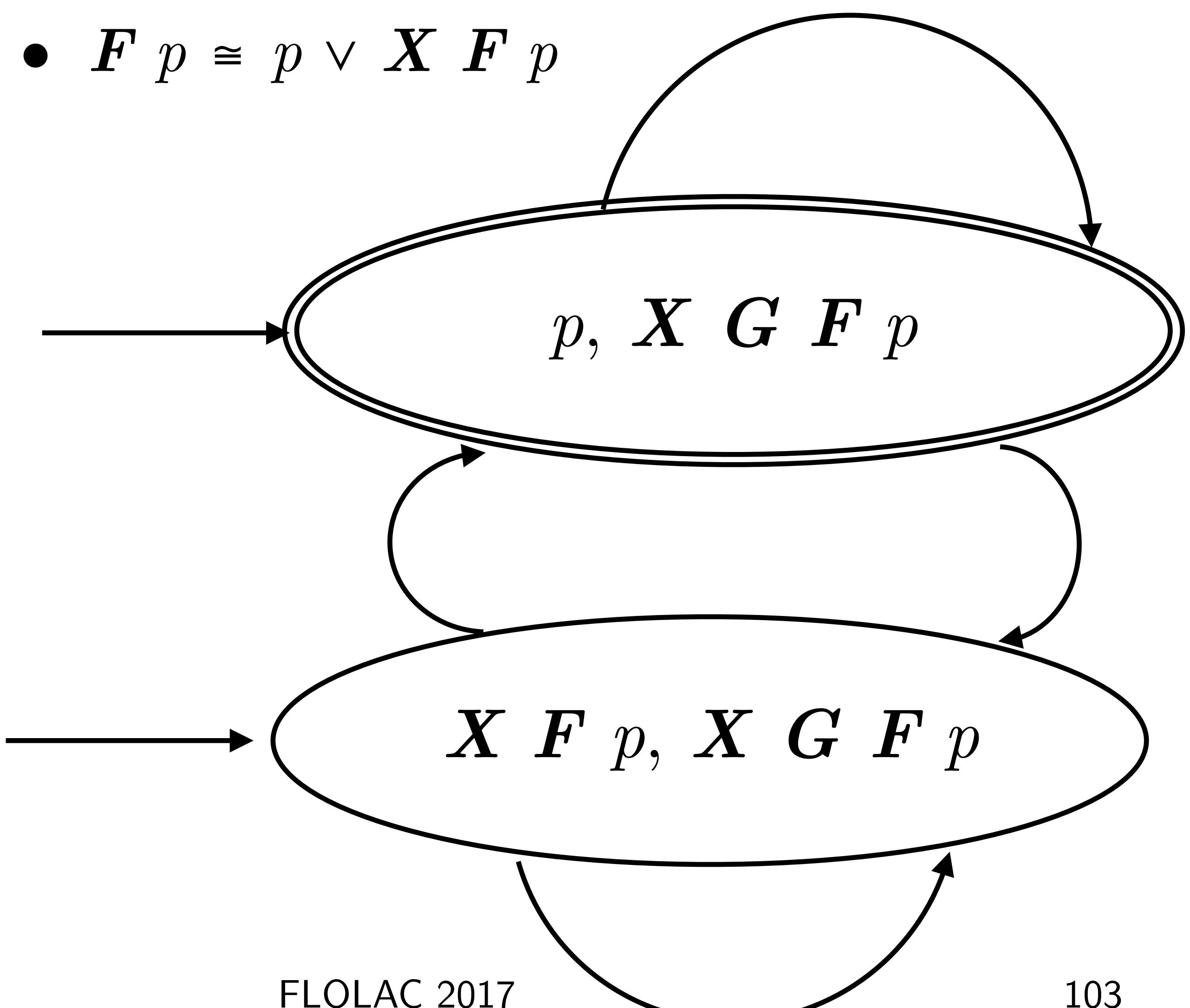
From LTL to Labeled NGW

- Translate an LTL formula *f* to a labeled NGW (with labels on states).
  - Take the *negation normal form* (NNF) of f.
  - Expand f_{NNF} into basic formulas as the initial states.
  - Construct successors of states based on $\boldsymbol{X}$-formulas.
  - For each subformula $g\ \boldsymbol{U}\ h$, create an acceptance set such that h will become true eventually.

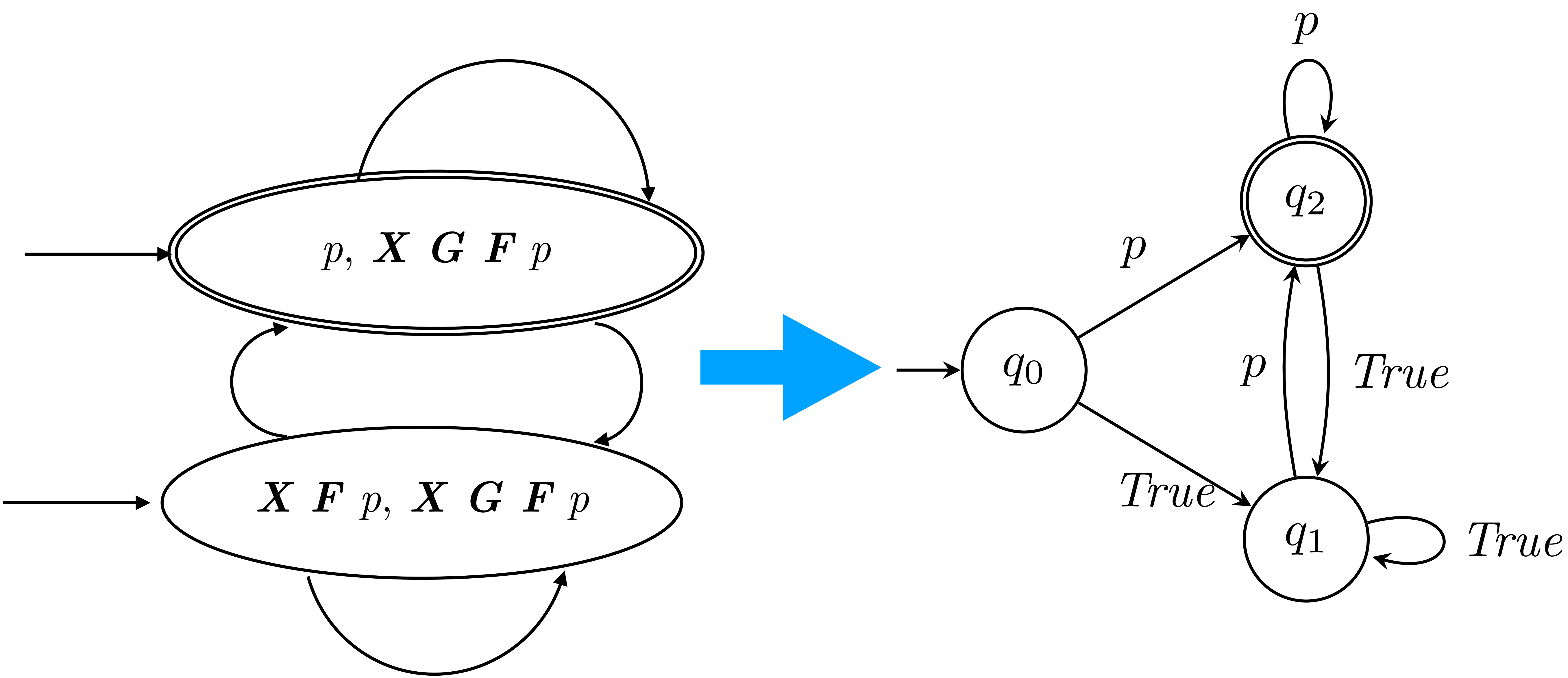
NNF: negation only occurs right before propositions

From LTL to Labeled NGW Example

- $f := \boldsymbol{G}\ \boldsymbol{F}\ p$
- $\boldsymbol{G}\ \boldsymbol{F}\ p \cong (p \vee \boldsymbol{X}\ \boldsymbol{F}\ p) \wedge \boldsymbol{X}\ \boldsymbol{G}\ \boldsymbol{F}\ p \cong (p \wedge \boldsymbol{X}\ \boldsymbol{G}\ \boldsymbol{F}\ p) \vee (\boldsymbol{X}\ \boldsymbol{F}\ p \wedge \boldsymbol{X}\ \boldsymbol{G}\ \boldsymbol{F}\ p)$
- $\boldsymbol{F}\ p \cong p \vee \boldsymbol{X}\ \boldsymbol{F}\ p$



From Labeled NGW to NGW



From NGW to NBW

- Apply the same technique in the intersection of NBW.
- Use an index i to remember the next acceptance set in $\{F_1, F_2, ..., F_n\}$ to be passed.
- Once a state in F_i is passed, increase the index i by 1.
- If every $F_i \in \{F_1, F_2, ..., F_\mathrm{n}\}$ has been passed at least once, change the index to 0 and set the index to 1 in the successors.
- A run is accepting if the index 0 is passed infinitely many times.

Tools

- LTL2BA (http://www.lsv.fr/~gastin/ltl2ba/index.php)
- LTL3BA (https://sourceforge.net/projects/ltl3ba/)
- SPIN (http://spinroot.com/spin/whatispin.html)
- NuSMV (http://nusmv.fbk.eu)
- GOAL (http://goal.im.ntu.edu.tw/wiki/doku.php)