COMP3630 / COMP6363

week 2: **Properties of Regular Languages**

This Lecture Covers Chapter 4 of HMU: Properties of Regular Languages

slides created by: Dirk Pattinson, based on material by
Peter Hoefner and Rob van Glabbeck; with improvements by Pascal Bercher

convenor & lecturer: Pascal Bercher

The Australian National University

Semester 1, 2023

Content of this Chapter

- Pumping Lemma for regular languages
- Some more properties of regular languages
- Decision properties of regular languages
- Equivalence and minimization of automata

Additional Reading: Chapter 4 of HMU.

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.
- What can we say if a language is defined by enumeration or by a predicate?

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.
- What can we say if a language is defined by enumeration or by a predicate?
  - Is $L = \{w \in \{0,1\}^* : w \text{ does not contain } 10\}$ regular?

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.
- What can we say if a language is defined by enumeration or by a predicate?
  - Is $L = \{w \in \{0,1\}^* : w \text{ does not contain } 10\}$ regular?
  - Is $L = \{0^n 1^n : n \geq 0\}$ regular?

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.
- What can we say if a language is defined by enumeration or by a predicate?
  - Is $L = \{w \in \{0,1\}^* : w \text{ does not contain } 10\}$ regular?
  - Is $L = \{0^n 1^n : n \geq 0\}$ regular?
  - How do we answer such questions without delving into each

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.
- What can we say if a language is defined by enumeration or by a predicate?
  - Is $L = \{w \in \{0,1\}^* : w \text{ does not contain } 10\}$ regular?
  - Is $L = \{0^n 1^n : n \geq 0\}$ regular?
  - How do we answer such questions without delving into each
- Is there an inherent structure to the strings belonging to a regular language?

Pumping Lemma

- We know: If a language is given by a regular expression, or a DFA, it is regular.
- What can we say if a language is defined by enumeration or by a predicate?
  - Is $L = \{w \in \{0,1\}^* : w \text{ does not contain } 10\}$ regular?
  - Is $L = \{0^n 1^n : n \geq 0\}$ regular?
  - How do we answer such questions without delving into each
- Is there an inherent structure to the strings belonging to a regular language?

Lemma 4.1.1 (Pumping Lemma for Regular Languages)

Let L be a regular language. There there exists an $n \in \mathbb{N}, n \geq 1$ (depending on L) such that for any string $w \in L$ with $|w| \geq n$, there exist strings x, y, z such that:
(1) $w = xyz$
(2) $|xy| \leq n$
(3) $|y| > 0$
(4) $xy^i z \in L$ for $i \in \mathbb{N} \cup \{0\}$

Proof of the Pumping Lemma

- Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept L, and let $n := |Q|$.

Proof of the Pumping Lemma

- Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept L, and let $n := |Q|$.
- The claim is vacuously true if L contains only strings of length $n - 1$ or less.

Proof of the Pumping Lemma

- Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept L, and let $n := |Q|$.
- The claim is vacuously true if L contains only strings of length $n - 1$ or less.
- Suppose L contains a string $w = s_1 \cdots s_k \in L$ with $|w| = k \geq n$.

Proof of the Pumping Lemma

- Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept L, and let $n := |Q|$.
- The claim is vacuously true if L contains only strings of length $n-1$ or less.
- Suppose L contains a string $w = s_1 \cdots s_k \in L$ with $|w| = k \geq n$.
- Then, there must be a sequence of transitions that move A from q_0 to some final state upon reading w.

$$\underbrace{q_0 = q_{i_0} \longrightarrow q_{i_1} \longrightarrow q_{i_2} \longrightarrow \cdots \longrightarrow q_{i_n}}_{n \text{ symbols and } n+1 \text{ states}} \longrightarrow \cdots \longrightarrow q_{i_k} \in F$$

Proof of the Pumping Lemma

- Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept L, and let $n := |Q|$.
- The claim is vacuously true if L contains only strings of length $n - 1$ or less.
- Suppose L contains a string $w = s_1 \cdots s_k \in L$ with $|w| = k \geq n$.
- Then, there must be a sequence of transitions that move A from q_0 to some final state upon reading w.

$$\underbrace{q_0 = q_{i_0} \longrightarrow q_{i_1} \longrightarrow q_{i_2} \longrightarrow \cdots \longrightarrow q_{i_n}}_{n \text{ symbols and } n+1 \text{ states}} \longrightarrow \cdots \longrightarrow q_{i_k} \in F$$

- **SOME** state must be visited (at least) twice. Let $q_{i_a} = q_{i_b}$ for $i_0 \leq i_a < i_b \leq i_n$.

$$q_0 = q_{i_0} \overbrace{\xrightarrow{s_1} \cdots \xrightarrow{s_{i_a}}}^{x \text{ is read}} {\color{blue}q_{i_a}} \overbrace{\xrightarrow{s_{i_a}+1} \cdots \xrightarrow{s_{i_b}}}^{y \text{ is read}} {\color{blue}q_{i_b}} \overbrace{\xrightarrow{s_{i_b}+1} \cdots \xrightarrow{s_{i_n}} q_{i_n} \xrightarrow{s_{i_n}+1} \cdots \xrightarrow{s_{i_k}}}^{z \text{ is read}} q_{i_k} \in F$$

Proof of the Pumping Lemma

- Let DFA $A = (Q, \Sigma, \delta, q_0, F)$ accept L, and let $n := |Q|$.
- The claim is vacuously true if L contains only strings of length $n-1$ or less.
- Suppose L contains a string $w = s_1 \cdots s_k \in L$ with $|w| = k \geq n$.
- Then, there must be a sequence of transitions that move A from q_0 to some final state upon reading w.

$$\underbrace{q_0 = q_{i_0} \longrightarrow q_{i_1} \longrightarrow q_{i_2} \longrightarrow \cdots \longrightarrow q_{i_n}}_{n \text{ symbols and } n+1 \text{ states}} \longrightarrow \cdots \longrightarrow q_{i_k} \in F$$

- **SOME** state must be visited (at least) twice. Let $q_{i_a} = q_{i_b}$ for $i_0 \leq i_a < i_b \leq i_n$.

$$q_0 = q_{i_0} \overbrace{\xrightarrow{s_1} \cdots \xrightarrow{s_{i_a}}}^{x \text{ is read}} q_{i_a} \overbrace{\xrightarrow{s_{i_a}+1} \cdots \xrightarrow{s_{i_b}}}^{y \text{ is read}} q_{i_b} \overbrace{\xrightarrow{s_{i_b}+1} \cdots \xrightarrow{s_{i_n}} q_{i_n} \xrightarrow{s_{i_n}+1} \cdots \xrightarrow{s_{i_k}}}^{z \text{ is read}} q_{i_k} \in F$$

- (4) holds since the path for xy^iz is derived from the above either by deleting the subpath between q_{i_a} and q_{i_b} or by repeating it. All such paths end in $q_{i_k} \in F$.

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.
  - Suppose it is. By the pumping lemma, there exists $k \geq 1$ such that any $w \in L$, $|w| \geq k$ can be split as $w = xyz, |y| \geq 1$ and $|xy| \leq k$ s.t. $xy^i z \in L$ for all $i \geq 0$.

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.
  - Suppose it is. By the pumping lemma, there exists $k \geq 1$ such that any $w \in L$, $|w| \geq k$ can be split as $w = xyz, |y| \geq 1$ and $|xy| \leq k$ s.t. $xy^i z \in L$ for all $i \geq 0$.
  - let's apply this to the string $w = 0^k 1^k \in L$:

$$\underbrace{0\ldots0}_{x}\underbrace{0\ldots0}_{y}\underbrace{0\ldots0\,1\ldots1}_{z}\quad\text{with}\quad\overbrace{0\ldots0\,0\ldots0\,0\ldots0}^{k}\overbrace{1\ldots1}^{k}$$

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.
  - Suppose it is. By the pumping lemma, there exists $k \geq 1$ such that any $w \in L$, $|w| \geq k$ can be split as $w = xyz, |y| \geq 1$ and $|xy| \leq k$ s.t. $xy^i z \in L$ for all $i \geq 0$.
  - let's apply this to the string $w = 0^k 1^k \in L$:

$$\mathrlap{\overbrace{}^{k}\overbrace{}^{k}}\underbrace{0\ldots0}_{x}\,\underbrace{0\ldots0}_{y}\,\underbrace{0\ldots0\,1\ldots1}_{z}$$

  - As $|xy| \leq k$, this means that $x = 0^i$ and $y = 0^j$, $z = 0^p 1^k$ with $i + j + p = k$.

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.
  - Suppose it is. By the pumping lemma, there exists $k \geq 1$ such that any $w \in L$, $|w| \geq k$ can be split as $w = xyz, |y| \geq 1$ and $|xy| \leq k$ s.t. $xy^i z \in L$ for all $i \geq 0$.
  - let's apply this to the string $w = 0^k 1^k \in L$:

$$\overbrace{\underbrace{0\ldots0}_{x}\underbrace{0\ldots0}_{y}0\ldots0}^{k}\overbrace{1\ldots1}^{k}$$

$$z = 0\ldots0\,1\ldots1$$

  - As $|xy| \leq k$, this means that $x = 0^i$ and $y = 0^j$, $z = 0^p 1^k$ with $i + j + p = k$.
  - By the pumping lemma, $xy^0 z \in L$ but $xy^0 z = 0^i 0^p y^k$ and $i + p \neq k$ as $j = |y| \geq 1$, contradiction.

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.
  - Suppose it is. By the pumping lemma, there exists $k \geq 1$ such that any $w \in L$, $|w| \geq k$ can be split as $w = xyz, |y| \geq 1$ and $|xy| \leq k$ s.t. $xy^i z \in L$ for all $i \geq 0$.
  - let's apply this to the string $w = 0^k 1^k \in L$:

$$\overbrace{\underbrace{0\ldots0}_{x}\,\underbrace{0\ldots0}_{y}\,\mathrlap{\underbrace{}_{z}}0\ldots0}^{k}\,\overbrace{1\ldots1}^{k}$$

  - As $|xy| \leq k$, this means that $x = 0^i$ and $y = 0^j$, $z = 0^p 1^k$ with $i + j + p = k$.
  - By the pumping lemma, $xy^0 z \in L$ but $xy^0 z = 0^i 0^p y^k$ and $i + p \neq k$ as $j = |y| \geq 1$, contradiction.
- $L = \{w \in \{0,1\}^* : |w| \text{ is a prime}\}$ is **not** regular.

Applications of the Pumping Lemma

Using the Pumping Lemma, we can show

- $L = \{0^n 1^n : n \geq 0\}$ is **not** regular.
  - Suppose it is. By the pumping lemma, there exists $k \geq 1$ such that any $w \in L$, $|w| \geq k$ can be split as $w = xyz, |y| \geq 1$ and $|xy| \leq k$ s.t. $xy^i z \in L$ for all $i \geq 0$.
  - let's apply this to the string $w = 0^k 1^k \in L$:

$$\overbrace{\underbrace{0\ldots0}_{x}\underbrace{0\ldots0}_{y}\rlap{$\underbrace{}_{z}$}0\ldots0}^{k}\overbrace{1\ldots1}^{k}$$

  - As $|xy| \leq k$, this means that $x = 0^i$ and $y = 0^j$, $z = 0^p 1^k$ with $i + j + p = k$.
  - By the pumping lemma, $xy^0 z \in L$ but $xy^0 z = 0^i 0^p y^k$ and $i + p \neq k$ as $j = |y| \geq 1$, contradiction.
- $L = \{w \in \{0,1\}^* : |w| \text{ is a prime}\}$ is **not** regular.
- $L = \{ww^R : w \in \{0,1\}^*\}$ is **not** regular. [$w^R = w$ read from right to left].

Additional Properties of Regular Languages

- We already know regular languages are closed under:
  - union, intersection, concatenation, Kleene-$*$ closure, and difference.

Additional Properties of Regular Languages

- We already know regular languages are closed under:
  - union, intersection, concatenation, Kleene-$*$ closure, and difference.
- We'll see three more operations under which regular languages are closed.

Additional Properties of Regular Languages

- We already know regular languages are closed under:
  - union, intersection, concatenation, Kleene-$*$ closure, and difference.
- We'll see three more operations under which regular languages are closed.
- Let L^R be the language obtained by reversing each string ($(01)^R = 10$)

Theorem 4.2.1

Let L be regular. Then $L^R := \{w^R : w \in L\}$ is also regular.

Additional Properties of Regular Languages

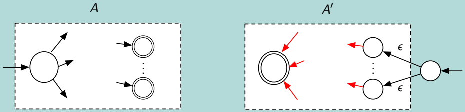
- We already know regular languages are closed under:
  - union, intersection, concatenation, Kleene-$*$ closure, and difference.
- We'll see three more operations under which regular languages are closed.
- Let L^R be the language obtained by reversing each string ($(01)^R = 10$)

Theorem 4.2.1

Let L be regular. Then $L^R := \{w^R : w \in L\}$ is also regular.

Proof of Theorem 4.2.1

- Let langauge L be accepted by DFA A.



- Let A' be the DFA obtained by: (a) Reversing each arrow in A; (b) swapping final and initial states; and (c) introduce ϵ-transitions to make initial state (of A') unique.
- Then L^R is accepted by A'.

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Proof of Theorem 4.2.2

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Proof of Theorem 4.2.2

- Let E be the regular expression corresponding to L

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Proof of Theorem 4.2.2

- Let E be the regular expression corresponding to L
- Let $h(E)$ be the expression obtained by replacing symbols $s \in \Sigma_1$ by $h(s)$.

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Proof of Theorem 4.2.2

- Let E be the regular expression corresponding to L
- Let $h(E)$ be the expression obtained by replacing symbols $s \in \Sigma_1$ by $h(s)$.
- Then $h(E)$ is a regular expression over Σ_2

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Proof of Theorem 4.2.2

- Let E be the regular expression corresponding to L
- Let $h(E)$ be the expression obtained by replacing symbols $s \in \Sigma_1$ by $h(s)$.
- Then $h(E)$ is a regular expression over Σ_2
- By a straightforward induction argument, we can show that $L(h(E)) = h(L(E))$

Closure under Homomorphisms

- A homomorphism is a map $h : \Sigma_1 \to \Sigma_2^*$.
- The map can be extended to strings by defining $s_1 \cdots s_k \overset{h}{\mapsto} h(s_1) \cdots h(s_k)$.

Theorem 4.2.2

Let L be regular. Then $h(L) := \{h(w) : w \in L\}$ is also regular.

Proof of Theorem 4.2.2

- Let E be the regular expression corresponding to L
- Let $h(E)$ be the expression obtained by replacing symbols $s \in \Sigma_1$ by $h(s)$.
- Then $h(E)$ is a regular expression over Σ_2
- By a straightforward induction argument, we can show that $L(h(E)) = h(L(E))$

Expressions Languages

$$\begin{tikzcd} E \arrow[r] \arrow[d, "h"'] & L(E) \arrow[d, "h"] & \text{(language over } \Sigma_1) \\ h(E) \arrow[r] & h(L(E)) = L(h(E)) & \text{(language over } \Sigma_2) \end{tikzcd}$$

Closure under Inverse Homomorphisms

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Proof of Theorem 4.2.3

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Proof of Theorem 4.2.3

- Let DFA $A = (Q, \Sigma_2, \delta, q_0, F)$ accept L

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Proof of Theorem 4.2.3

- Let DFA $A = (Q, \Sigma_2, \delta, q_0, F)$ accept L
- Let DFA $B = (Q, \Sigma_1, \gamma, q_0, F)$ where

$$\gamma(q, s) = \hat{\delta}(q, h(s))$$

[Depending on the input B mimics none, one, or many transitions of A]

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Proof of Theorem 4.2.3

- Let DFA $A = (Q, \Sigma_2, \delta, q_0, F)$ accept L
- Let DFA $B = (Q, \Sigma_1, \gamma, q_0, F)$ where

$$\gamma(q, s) = \hat{\delta}(q, h(s))$$

  [Depending on the input B mimics none, one, or many transitions of A]
- By definition, $\epsilon \in L(A)$ iff $q_0 \in F$ iff $\epsilon \in L(B)$

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Proof of Theorem 4.2.3

- Let DFA $A = (Q, \Sigma_2, \delta, q_0, F)$ accept L
- Let DFA $B = (Q, \Sigma_1, \gamma, q_0, F)$ where

$$\gamma(q, s) = \hat{\delta}(q, h(s))$$

  [Depending on the input B mimics none, one, or many transitions of A]
- By definition, $\epsilon \in L(A)$ iff $q_0 \in F$ iff $\epsilon \in L(B)$
- By induction, we can show that

$$s_1 \cdots s_k \in L(B) \Leftrightarrow h(s_1) \cdots h(s_k) \in L(A) = L$$

Closure under Inverse Homomorphisms

Theorem 4.2.3

Let L be regular. Then $h^{-1}(L) := \{w : h(w) \in L\}$ is also regular.

Proof of Theorem 4.2.3

- Let DFA $A = (Q, \Sigma_2, \delta, q_0, F)$ accept L
- Let DFA $B = (Q, \Sigma_1, \gamma, q_0, F)$ where

$$\gamma(q, s) = \hat{\delta}(q, h(s))$$

  [Depending on the input B mimics none, one, or many transitions of A]
- By definition, $\epsilon \in L(A)$ iff $q_0 \in F$ iff $\epsilon \in L(B)$
- By induction, we can show that

$$s_1 \cdots s_k \in L(B) \Leftrightarrow h(s_1) \cdots h(s_k) \in L(A) = L$$

- Hence, B accepts $h^{-1}(L)$.

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?
  - Is the language accepted by a DFA non-empty?

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?
  - Is the language accepted by a DFA non-empty?
  - Does the language accepted by a DFA contain a given string w?

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?
  - Is the language accepted by a DFA non-empty?
  - Does the language accepted by a DFA contain a given string w?
  - Is the language accepted by a DFA infinite?

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?
  - Is the language accepted by a DFA non-empty?
  - Does the language accepted by a DFA contain a given string w?
  - Is the language accepted by a DFA infinite?
  - Do two given DFAs accept the same language?

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?
  - Is the language accepted by a DFA non-empty?
  - Does the language accepted by a DFA contain a given string w?
  - Is the language accepted by a DFA infinite?
  - Do two given DFAs accept the same language?
  - Given two DFAs A and B, is $L(A) \subseteq L(B)$?

Decision Properties

- DFAs and regular expressions are **finite representations** of regular languages
- How do we ascertain if a particular property is satisfied by a language?
  - Is the language accepted by a DFA non-empty?
  - Does the language accepted by a DFA contain a given string w?
  - Is the language accepted by a DFA infinite?
  - Do two given DFAs accept the same language?
  - Given two DFAs A and B, is $L(A) \subseteq L(B)$?
- We will look at the above 5 questions assuming that regular languages are defined by DFAs. (If the language is specified by an expression, we can convert it to a DFA!)

Decision Properties (Emptiness, Membership, Infiniteness)

- **Emptiness:** If one is given a DFA with n states that accepts L, we can find all the states reachable from the initial state in $O(n^2)$ time. If no final state is reachable, L must be empty.

Decision Properties (Emptiness, Membership, Infiniteness)

- **Emptiness:** If one is given a DFA with n states that accepts L, we can find all the states reachable from the initial state in $O(n^2)$ time. If no final state is reachable, L must be empty.
- **Membership:** If one is given a DFA with n states that accepts L, given string w, we can simply identify the transitions corresponding to w one symbol at a time. If the last state is an accepting state, then w must be in the language. This takes no more than $O(|w|)$ time steps.

Decision Properties (Emptiness, Membership, Infiniteness)

- **Emptiness:** If one is given a DFA with n states that accepts L, we can find all the states reachable from the initial state in $O(n^2)$ time. If no final state is reachable, L must be empty.
- **Membership:** If one is given a DFA with n states that accepts L, given string w, we can simply identify the transitions corresponding to w one symbol at a time. If the last state is an accepting state, then w must be in the language. This takes no more than $O(|w|)$ time steps.
- **Infiniteness:** We can reduce the problem of infiniteness to finding a cycle in the directed graph (a.k.a. transition diagram) of the DFA.

Decision Properties (Emptiness, Membership, Infiniteness)

- **Emptiness:** If one is given a DFA with n states that accepts L, we can find all the states reachable from the initial state in $O(n^2)$ time. If no final state is reachable, L must be empty.
- **Membership:** If one is given a DFA with n states that accepts L, given string w, we can simply identify the transitions corresponding to w one symbol at a time. If the last state is an accepting state, then w must be in the language. This takes no more than $O(|w|)$ time steps.
- **Infiniteness:** We can reduce the problem of infiniteness to finding a cycle in the directed graph (a.k.a. transition diagram) of the DFA.
  - First, delete any node unreachable from the initial node ($O(n^2)$ complexity).
  - Next, delete nodes that cannot reach any final node ($O(n^3)$ complexity).
  - Use depth-first search (DFS) to find a cycle in the remaining graph ($O(n^2)$ complexity).

Decision Properties (Emptiness, Membership, Infiniteness)

- **Emptiness:** If one is given a DFA with n states that accepts L, we can find all the states reachable from the initial state in $O(n^2)$ time. If no final state is reachable, L must be empty.
- **Membership:** If one is given a DFA with n states that accepts L, given string w, we can simply identify the transitions corresponding to w one symbol at a time. If the last state is an accepting state, then w must be in the language. This takes no more than $O(|w|)$ time steps.
- **Infiniteness:** We can reduce the problem of infiniteness to finding a cycle in the directed graph (a.k.a. transition diagram) of the DFA.
  - First, delete any node unreachable from the initial node ($O(n^2)$ complexity).
  - Next, delete nodes that cannot reach any final node ($O(n^3)$ complexity).
  - Use depth-first search (DFS) to find a cycle in the remaining graph ($O(n^2)$ complexity).
- **Q:** How do runtimes change if we have an NFA?

Decision Properties (Equivalence)

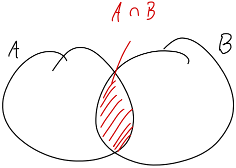
- **Equivalence:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) = L(B)$?

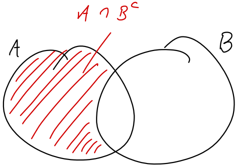
Decision Properties (Equivalence)

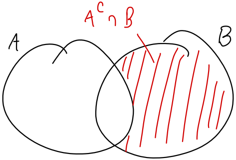
- **Equivalence:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) = L(B)$?

$$L(A) = L(B) \Leftrightarrow \begin{array}{l} L(A) \cap L(B)^c = \emptyset \\ L(A)^c \cap L(B) = \emptyset \end{array}$$

Why is this true?







Thus:

$A \cap B = A = B$ iff

$A \cap B^c = A^c \cap B = \emptyset$

Decision Properties (Equivalence)

- **Equivalence:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) = L(B)$?

$$L(A) = L(B) \Leftrightarrow \begin{array}{l} L(A) \cap L(B)^c = \emptyset \\ L(A)^c \cap L(B) = \emptyset \end{array}$$

Run A and B in parallel. (Not their complement-versions!)

  - $L(A) \cap L(B)^c$: Accept if resp. paths end in F_A and F_B^c.
  - $L(A)^c \cap L(B)$: Accept if resp. paths end in F_A^c and F_B.

Decision Properties (Equivalence)

- **Equivalence:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) = L(B)$?

$$L(A) = L(B) \Leftrightarrow \begin{array}{l} L(A) \cap L(B)^c = \emptyset \\ L(A)^c \cap L(B) = \emptyset \end{array}$$

Run A and B in parallel. (Not their complement-versions!)

  - $L(A) \cap L(B)^c$: Accept if resp. paths end in F_A and F_B^c.
  - $L(A)^c \cap L(B)$: Accept if resp. paths end in F_A^c and F_B.

- Use **product** DFA: Construct $C = (Q_C, \Sigma, \delta_C, q_{C0}, F_C)$ defined by

$$\begin{aligned} Q_C &= Q_A \times Q_B && \text{[Cartesian Product]} \\ q_{C0} &= (q_{A0}, q_{B0}) && \\ \delta_C((q, q'), s) &= (\delta_A(q, s), \delta_B(q', s)) && \text{[Both DFAs are simulated in parallel]} \\ F_C &= (F_A \times F_B^c) \cup (F_A^c \times F_B) && \text{[accept strings in exactly one of } L(A) \text{ or } L(B)\text{]} \end{aligned}$$

Decision Properties (Equivalence)

- **Equivalence:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) = L(B)$?

$$L(A) = L(B) \Leftrightarrow \begin{array}{l} L(A) \cap L(B)^c = \emptyset \\ L(A)^c \cap L(B) = \emptyset \end{array}$$

Run A and B in parallel. (Not their complement-versions!)

  - $L(A) \cap L(B)^c$: Accept if resp. paths end in F_A and F_B^c.
  - $L(A)^c \cap L(B)$: Accept if resp. paths end in F_A^c and F_B.

- Use **product** DFA: Construct $C = (Q_C, \Sigma, \delta_C, q_{C0}, F_C)$ defined by

$$\begin{aligned} Q_C &= Q_A \times Q_B && \text{[Cartesian Product]} \\ q_{C0} &= (q_{A0}, q_{B0}) \\ \delta_C((q, q'), s) &= (\delta_A(q, s), \delta_B(q', s)) && \text{[Both DFAs are simulated in parallel]} \\ F_C &= (F_A \times F_B^c) \cup (F_A^c \times F_B) && \text{[accept strings in exactly one of } L(A) \text{ or } L(B)\text{]} \end{aligned}$$

$$L(A) = L(B) \Leftrightarrow L(C) = \emptyset$$

(and you know how to check for an empty language of a DFA!)

Decision Properties (Inclusion)

- **Inclusion:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) \subseteq L(B)$?

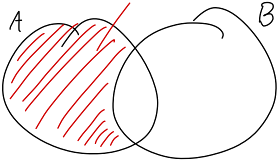
Decision Properties (Inclusion)

- **Inclusion:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) \subseteq L(B)$?

$$L(A) \subseteq L(B) \Leftrightarrow \ L(A) \cap L(B)^c = \emptyset$$

Why is this true?

$A \cap B^c = \emptyset$



Thus: $A \subseteq B$

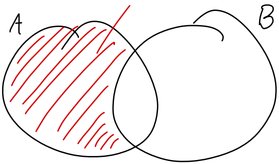
Decision Properties (Inclusion)

- **Inclusion:** Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$, how do we ascertain if $L(A) \subseteq L(B)$?

$$L(A) \subseteq L(B) \Leftrightarrow \; L(A) \cap L(B)^c = \emptyset$$

Why is this true?

$A \cap B^c = \emptyset$

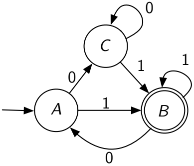


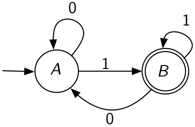
Thus: $A \subseteq B$

What now? Everything as before! Only now $F_C = F_A \times F_B^c$.

DFA State Minmimization

- Given two DFAs, we know how to test if they accept the same language.
- Is there a unique minimal DFA for a given regular language?
- Given a DFA, can we **reduce** the number of states without altering the language it accepts?





Clearly, the two DFAs accept the same language and state C is unnecessary.

- How do we (identify and) remove 'unnecessary' states without altering the underlying language?

DFA State Minimimization

- State minimization requires a notion of **equivalence** or **distinguishability** of states.

DFA State Minimimization

- State minimization requires a notion of **equivalence** or **distinguishability** of states.
- Clearly, **distinguishability** of two states must be based on **finality**

DFA State Minimimization

- State minimization requires a notion of **equivalence** or **distinguishability** of states.
- Clearly, **distinguishability** of two states must be based on **finality**

states p and q are **equivalent** or indistinguishable $\Leftrightarrow$ $\hat{\delta}(p, w) \in F$ whenever $\hat{\delta}(q, w) \in F$ for every $w \in \Sigma^*$.

DFA State Minimimization

- State minimization requires a notion of **equivalence** or **distinguishability** of states.
- Clearly, **distinguishability** of two states must be based on **finality**

states p and q are **equivalent** or indistinguishable $\Leftrightarrow$ $\hat{\delta}(p, w) \in F$ whenever $\hat{\delta}(q, w) \in F$ for every $w \in \Sigma^*$.

states p and q are **distinguishable** $\Leftrightarrow$ exactly one of $\hat{\delta}(p, w)$ or $\hat{\delta}(q, w)$ is in F for some $w \in \Sigma^*$.

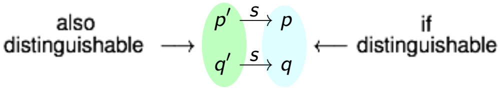
DFA State Minimimization

- State minimization requires a notion of **equivalence** or **distinguishability** of states.
- Clearly, **distinguishability** of two states must be based on **finality**

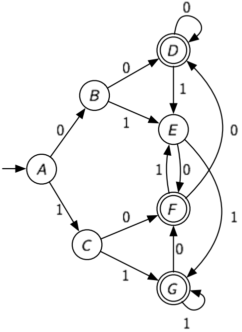
states p and q are **equivalent** or indistinguishable $\Leftrightarrow$ $\hat{\delta}(p, w) \in F$ whenever $\hat{\delta}(q, w) \in F$ for every $w \in \Sigma^*$.

states p and q are **distinguishable** $\Leftrightarrow$ exactly one of $\hat{\delta}(p, w)$ or $\hat{\delta}(q, w)$ is in F for some $w \in \Sigma^*$.

- **Table Filling** Algorithm identifies equivalent and distinguishable pairs of states.
  - Any final state is **distinguishable** from a non-final state (and vice versa)
  - If (a) p and q are distinguishable; and there exist states p', q', and symbol s such that (b) $\hat{\delta}(p', s) = p$ and (c) $\hat{\delta}(q', s) = q$, then p' and q' are also distinguishable

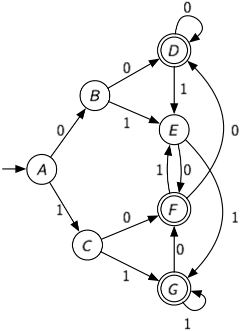


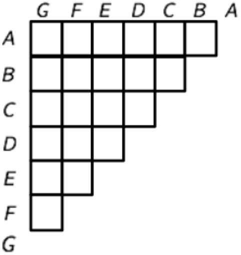
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

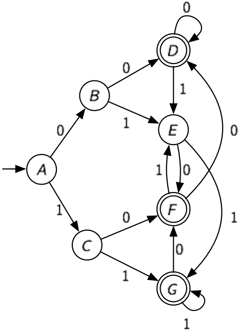
Identifying pairs of (In)distinguishable States: An Example





- Fill in $\times$ whenever one component of pair is final, and other is not.

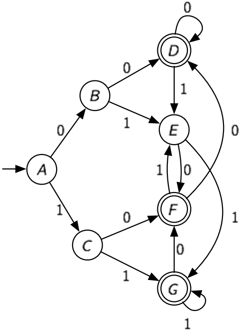
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

- Fill in $\times$ whenever one component of pair is final, and other is not.
- Fill in $\times$ if 1 moves the pair of states to a distinguishable pair

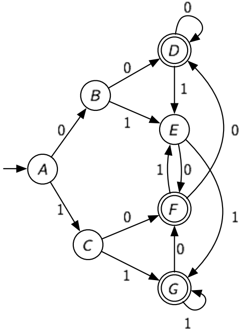
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

- Fill in × whenever one component of pair is final, and other is not.
- Fill in × if 1 moves the pair of states to a distinguishable pair
- Fill in × if 0 moves the pair of states to a distinguishable pair

Identifying pairs of (In)distinguishable States: An Example

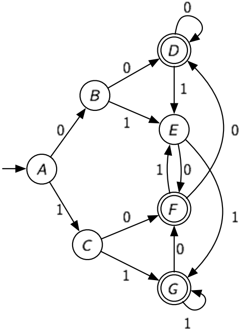


	G	F	E	D	C	B	A
A							
B							
C							
D							
E							
F							
G							

- Fill in $\times$ whenever one component of pair is final, and other is not.
- Fill in $\times$ if 1 moves the pair of states to a distinguishable pair
- Fill in $\times$ if 0 moves the pair of states to a distinguishable pair
- Repeat until no progress

(This slide is added to the handout so you can try it yourself!)

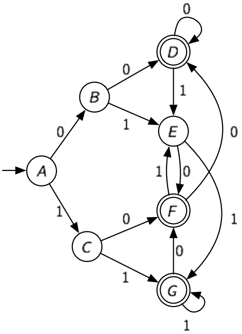
Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A	×	×	×	×	×	×	
B	×	×	×	×	×		
C	×	×		×			
D	×		×				
E	×	×					
F	×						
G							

- Fill in × whenever one component of pair is final, and other is not.
- Fill in × if 1 moves the pair of states to a distinguishable pair
- Fill in × if 0 moves the pair of states to a distinguishable pair
- Repeat until no progress

Identifying pairs of (In)distinguishable States: An Example



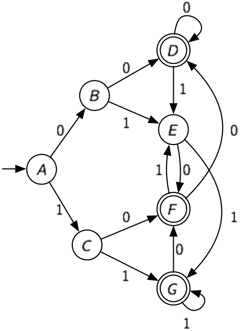
	G	F	E	D	C	B	A
A	×	×	×	×	×	×	
B	×	×	×	×	×		
C	×	×		×			
D	×		×				
E	×	×					
F	×						
G							

- Fill in $\times$ whenever one component of pair is final, and other is not.
- Fill in $\times$ if 1 moves the pair of states to a distinguishable pair
- Fill in $\times$ if 0 moves the pair of states to a distinguishable pair
- Repeat until no progress

Theorem 4.4.1

Any two states without a $\times$ sign are equivalent.

Identifying pairs of (In)distinguishable States: An Example



	G	F	E	D	C	B	A
A	×	×	×	×	×	×	
B	×	×	×	×	×		
C	×	×		×			
D	×		×				
E	×	×					
F	×						
G							

- Fill in × whenever one component of pair is final, and other is not.
- Fill in × if 1 moves the pair of states to a distinguishable pair
- Fill in × if 0 moves the pair of states to a distinguishable pair
- Repeat until no progress

Theorem 4.4.1

Any two states without a × sign are equivalent.

- Proof idea: If two states are distinguishable, the algorithm **will** fill a × eventually.

Table-filling Algorithm

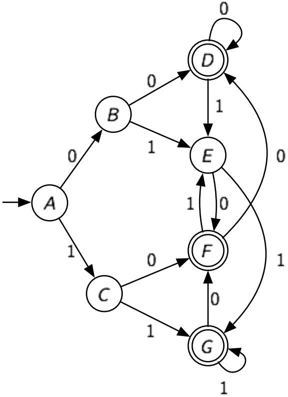
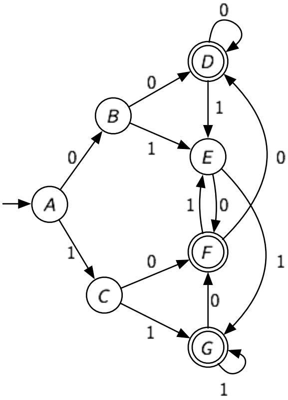
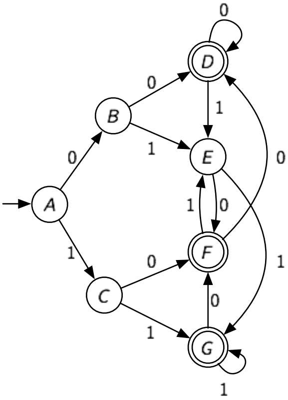


Table-filling Algorithm



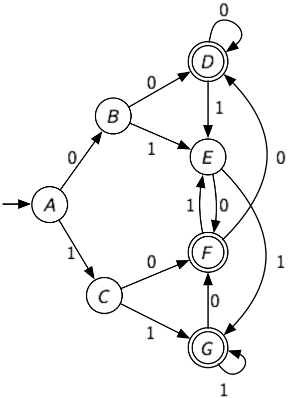
- Delete states not reachable from start states

Table-filling Algorithm



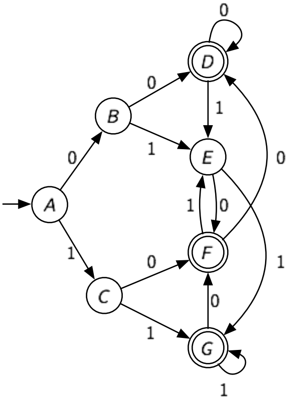
- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state

Table-filling Algorithm



- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state
- Find distinguishable and equivalent pairs of states

Table-filling Algorithm

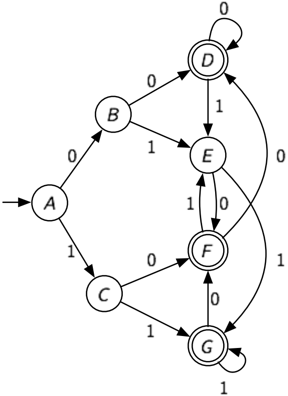


- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state
- Find distinguishable and equivalent pairs of states
- Find equivalence classes of indistinguishable states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$

	G	F	E	D	C	B	A
A	×	×	×	×	×	×	
B	×	×	×	×	×		
C	×	×		×			
D	×		×				
E	×	×					
F	×						
G							

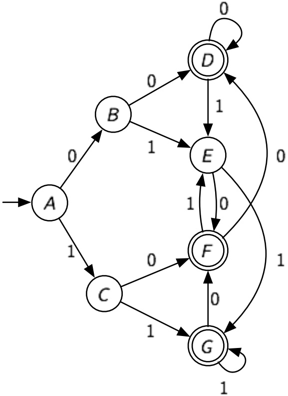
Color-blind?
red: A/B, blue: A/E, A/C, B/E, B/C, D/G, F/G

Table-filling Algorithm



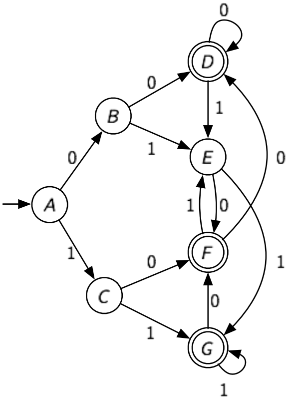
- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state
- Find distinguishable and equivalent pairs of states
- Find equivalence classes of indistinguishable states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a state

Table-filling Algorithm



- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state
- Find distinguishable and equivalent pairs of states
- Find equivalence classes of indistinguishable states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a state
- Delete parallel transitions with same label.

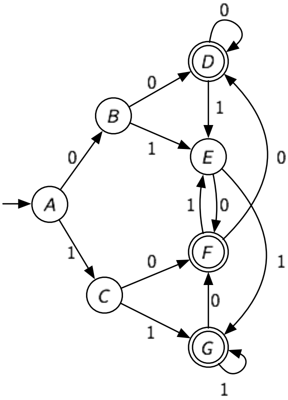
Table-filling Algorithm



- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state
- Find distinguishable and equivalent pairs of states
- Find equivalence classes of indistinguishable states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a state
- Delete parallel transitions with same label.

Remark: The resultant transition diagram will be a DFA.

Table-filling Algorithm



- Delete states not reachable from start states
- Delete any non-starting state that cannot reach any final state
- Find distinguishable and equivalent pairs of states
- Find equivalence classes of indistinguishable states. In this example: $\{A\}, \{B\}, \{C, E\}, \{D, F\}, \{G\}$
- Collapse each equivalence class of states to a state
- Delete parallel transitions with same label.

Remark: The resultant transition diagram will be a DFA.

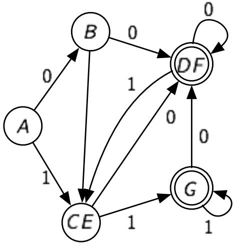


Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
    - Rename states in Q_B so that Q_A and Q_B are disjoint.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
    - Rename states in Q_B so that Q_A and Q_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are 2 start states)

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
    - Rename states in Q_B so that Q_A and Q_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are 2 start states)
    - Run table-filling on $Q_A \cup Q_B$.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
    - Rename states in Q_B so that Q_A and Q_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are 2 start states)
    - Run table-filling on $Q_A \cup Q_B$.
    - q_{A0} and q_{B0} are indistinguishable $\Leftrightarrow L(A) = L(B)$.

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
    - Rename states in Q_B so that Q_A and Q_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are 2 start states)
    - Run table-filling on $Q_A \cup Q_B$.
    - q_{A0} and q_{B0} are indistinguishable $\Leftrightarrow L(A) = L(B)$.

  [Why?] If w distinguishes q_{A0} from q_{B0} then w cannot be in both $L(A)$ and $L(B)$

Table-filling: Other Uses

- Test equivalence of languages accepted by 2 DFAs.
  - Given $A = (Q_A, \Sigma, \delta_A, q_{A0}, F_A)$ and $B = (Q_B, \Sigma, \delta_B, q_{B0}, F_B)$:
    - Rename states in Q_B so that Q_A and Q_B are disjoint.
    - View A and B together as one DFA
      (Ignore the fact that there are 2 start states)
    - Run table-filling on $Q_A \cup Q_B$.
    - q_{A0} and q_{B0} are indistinguishable $\Leftrightarrow L(A) = L(B)$.

  [Why?] If w distinguishes q_{A0} from q_{B0} then w cannot be in both $L(A)$ and $L(B)$
- Suppose a DFA A cannot be minimized further by table-filling. Then, A has the least number of states among all DFAs that accept $L(A)$