State Minimization

CENG 280

Course outline

- Preliminaries: Alphabets and languages
- Regular languages
  - Regular expressions
  - Finite automata: DFA and NFA
  - Finite automata - regular languages
  - Languages that are and that are not regular, Pumping lemma
  - State minimization for DFA
- Context-free languages
- Turing-machines

State Minimization

- Equivalence relations induced by language and automata
- MyHill-Nerode theorem
- State minimization algorithm

State Minimization

- FA are useful to design algorithms and recognize RL.

State Minimization

- FA are useful to design algorithms and recognize RL.
- Not powerful to compare the number of occurrences of symbols in a string (no writable memory)

State Minimization

- FA are useful to design algorithms and recognize RL.
- Not powerful to compare the number of occurrences of symbols in a string (no writable memory)
- Still a useful computation machine, and it is important to reduce the number of states for computational efficiency.

State Minimization

- FA are useful to design algorithms and recognize RL.
- Not powerful to compare the number of occurrences of symbols in a string (no writable memory)
- Still a useful computation machine, and it is important to reduce the number of states for computational efficiency.
- Given M with k states, can you write a machine M' with $k < k'$ states such that $M = M'$?

- FA are useful to design algorithms and recognize RL.
- Not powerful to compare the number of occurrences of symbols in a string (no writable memory)
- Still a useful computation machine, and it is important to reduce the number of states for computational efficiency.
- Given M with k states, can you write a machine M' with $k < k'$ states such that $M = M'$?
- How?

State Minimization

- FA are useful to design algorithms and recognize RL.
- Not powerful to compare the number of occurrences of symbols in a string (no writable memory)
- Still a useful computation machine, and it is important to reduce the number of states for computational efficiency.
- Given M with k states, can you write a machine M' with $k < k'$ states such that $M = M'$?
- How?
  - Remove unreachable states.

State Minimization

- FA are useful to design algorithms and recognize RL.
- Not powerful to compare the number of occurrences of symbols in a string (no writable memory)
- Still a useful computation machine, and it is important to reduce the number of states for computational efficiency.
- Given M with k states, can you write a machine M' with $k < k'$ states such that $M = M'$?
- How?
  - Remove unreachable states.
  - Merge equivalent states.

Equivalence for L

Definition (language)

Let $L \subseteq \Sigma^\star$ be a language, and let $x, y \in \Sigma^\star$. x and y are said to be equivalent with respect to L, if for all $z \in \Sigma^\star$, $xz \in L$ iff $yz \in L$. The equivalent strings are denoted by $x \approx_L y$. Note that $\approx_L$ is an equivalence relation.

Equivalence for L

Definition (language)

Let $L \subseteq \Sigma^\star$ be a language, and let $x, y \in \Sigma^\star$. x and y are said to be equivalent with respect to L, if for all $z \in \Sigma^\star$, $xz \in L$ iff $yz \in L$. The equivalent strings are denoted by $x \approx_L y$. Note that $\approx_L$ is an equivalence relation.

If $x \approx_L y$, then either $x, y \in L$ or $x, y \notin L$ (why?).

Equivalence for L

Definition (language)

Let $L \subseteq \Sigma^\star$ be a language, and let $x, y \in \Sigma^\star$. x and y are said to be equivalent with respect to L, if for all $z \in \Sigma^\star$, $xz \in L$ iff $yz \in L$. The equivalent strings are denoted by $x \approx_L y$. Note that $\approx_L$ is an equivalence relation.

If $x \approx_L y$, then either $x, y \in L$ or $x, y \notin L$ (why?).
If $x \approx_L y$, then appending the same string z to both of them results in two strings that are either both in L or both not in L.

Equivalence for L

Definition (language)

Let $L \subseteq \Sigma^\star$ be a language, and let $x, y \in \Sigma^\star$. x and y are said to be equivalent with respect to L, if for all $z \in \Sigma^\star$, $xz \in L$ iff $yz \in L$. The equivalent strings are denoted by $x \approx_L y$. Note that $\approx_L$ is an equivalence relation.

If $x \approx_L y$, then either $x, y \in L$ or $x, y \notin L$ (why?).
If $x \approx_L y$, then appending the same string z to both of them results in two strings that are either both in L or both not in L.

Example

Equivalence classes of $\mathcal{L}(a^\star)$?

Equivalence for L

Definition (language)

Let $L \subseteq \Sigma^\star$ be a language, and let $x, y \in \Sigma^\star$. x and y are said to be equivalent with respect to L, if for all $z \in \Sigma^\star$, $xz \in L$ iff $yz \in L$. The equivalent strings are denoted by $x \approx_L y$. Note that $\approx_L$ is an equivalence relation.

If $x \approx_L y$, then either $x, y \in L$ or $x, y \notin L$ (why?).
If $x \approx_L y$, then appending the same string z to both of them results in two strings that are either both in L or both not in L.

Example

Equivalence classes of $\mathcal{L}(a^\star)$?
Equivalence classes of $\mathcal{L}((ab \cup ba)^\star)$?

Equivalence for L

Definition (language)

Let $L \subseteq \Sigma^\star$ be a language, and let $x, y \in \Sigma^\star$. x and y are said to be equivalent with respect to L, if for all $z \in \Sigma^\star$, $xz \in L$ iff $yz \in L$. The equivalent strings are denoted by $x \approx_L y$. Note that $\approx_L$ is an equivalence relation.

If $x \approx_L y$, then either $x, y \in L$ or $x, y \notin L$ (why?).
If $x \approx_L y$, then appending the same string z to both of them results in two strings that are either both in L or both not in L.

Example

Equivalence classes of $\mathcal{L}(a^\star)$?
Equivalence classes of $\mathcal{L}((ab \cup ba)^\star)$?

$\approx_L$ form a partition of $\Sigma^\star$

Equivalence for M

Definition (machine)

Let $M = (K, \Sigma, \delta, s, F)$ be a deterministic finite automaton. Two strings $x, y \in \Sigma^\star$ are equivalent with respect to M, denoted by $x \sim_M y$, if there exists $q \in K$ such that $(s, x) \vdash_M^\star (q, e)$ and $(s, y) \vdash_M^\star (q, e)$.

Equivalence for M

Definition (machine)

Let $M = (K, \Sigma, \delta, s, F)$ be a deterministic finite automaton. Two strings $x, y \in \Sigma^\star$ are equivalent with respect to M, denoted by $x \sim_M y$, if there exists $q \in K$ such that $(s, x) \vdash_M^\star (q, e)$ and $(s, y) \vdash_M^\star (q, e)$.

Example

Consider M for $\mathcal{L}((ab \cup ba)^\star)$ and $\mathcal{L}(a^\star)$.

Equivalence for M

Definition (machine)

Let $M = (K, \Sigma, \delta, s, F)$ be a deterministic finite automaton. Two strings $x, y \in \Sigma^\star$ are equivalent with respect to M, denoted by $x \sim_M y$, if there exists $q \in K$ such that $(s, x) \vdash_M^\star (q, e)$ and $(s, y) \vdash_M^\star (q, e)$.

Example

Consider M for $\mathcal{L}((ab \cup ba)^\star)$ and $\mathcal{L}(a^\star)$.

Both of the equivalence relations are over $\Sigma^\star$, and each of them form a partition of it.

Equivalence for M

Definition (machine)

Let $M = (K, \Sigma, \delta, s, F)$ be a deterministic finite automaton. Two strings $x, y \in \Sigma^\star$ are equivalent with respect to M, denoted by $x \sim_M y$, if there exists $q \in K$ such that $(s, x) \vdash_M^\star (q, e)$ and $(s, y) \vdash_M^\star (q, e)$.

Example

Consider M for $\mathcal{L}((ab \cup ba)^\star)$ and $\mathcal{L}(a^\star)$.

Both of the equivalence relations are over $\Sigma^\star$, and each of them form a partition of it.

Theorem

For any deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$, and any strings $x, y \in \Sigma^\star$, if $x \sim_M y$ then $x \approx_{L(M)} y$.

$\sim_M$ and $\approx_L$

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

$\sim_M$ and $\approx_L$

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

- If $\sim$ is a refinement of $\approx$, then each equivalence class of $\sim$ is contained in an equivalence class of $\approx$.

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

- If $\sim$ is a refinement of $\approx$, then each equivalence class of $\sim$ is contained in an equivalence class of $\approx$.
- In other words, each equivalence class of $\approx$ is union of one or more equivalence classes of $\sim$. (example?)

$\sim_M$ and $\approx_L$

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

- If $\sim$ is a refinement of $\approx$, then each equivalence class of $\sim$ is contained in an equivalence class of $\approx$.
- In other words, each equivalence class of $\approx$ is union of one or more equivalence classes of $\sim$. (example?)
- Theorem 5 implies that $\sim_M$ is a refinement of $\approx_{L(M)}$.

$\sim_M$ and $\approx_L$

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

- If $\sim$ is a refinement of $\approx$, then each equivalence class of $\sim$ is contained in an equivalence class of $\approx$.
- In other words, each equivalence class of $\approx$ is union of one or more equivalence classes of $\sim$. (example?)
- Theorem 5 implies that $\sim_M$ is a refinement of $\approx_{L(M)}$.
- Furthermore, we can deduce that for any deterministic automaton M, the number of states any automaton $\bar{M}$ that accepts $L(M)$ must be at least as large as the number of equivalence classes of $L(M)$ under $\approx_{L(M)}$.

$\sim_M$ and $\approx_L$

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

- If $\sim$ is a refinement of $\approx$, then each equivalence class of $\sim$ is contained in an equivalence class of $\approx$.
- In other words, each equivalence class of $\approx$ is union of one or more equivalence classes of $\sim$. (example?)
- Theorem 5 implies that $\sim_M$ is a refinement of $\approx_{L(M)}$.
- Furthermore, we can deduce that for any deterministic automaton M, the number of states any automaton $\bar{M}$ that accepts $L(M)$ must be at least as large as the number of equivalence classes of $L(M)$ under $\approx_{L(M)}$.
- Thus the number of equivalence classes of $\approx_{L(M)}$ gives a lower bound on the number of states of an automaton $\bar{M}$ that accepts $L(M)$.

$\sim_M$ and $\approx_L$

Definition (refinement)

An equivalence relation $\sim$ is a refinement of another $\approx$, if for all x, y, $x \sim y$ implies $x \approx y$.

- If $\sim$ is a refinement of $\approx$, then each equivalence class of $\sim$ is contained in an equivalence class of $\approx$.
- In other words, each equivalence class of $\approx$ is union of one or more equivalence classes of $\sim$. (example?)
- Theorem 5 implies that $\sim_M$ is a refinement of $\approx_{L(M)}$.
- Furthermore, we can deduce that for any deterministic automaton M, the number of states any automaton $\bar{M}$ that accepts $L(M)$ must be at least as large as the number of equivalence classes of $L(M)$ under $\approx_{L(M)}$.
- Thus the number of equivalence classes of $\approx_{L(M)}$ gives a lower bound on the number of states of an automaton $\bar{M}$ that accepts $L(M)$.
- Myhill-Nerode Theorem shows that this lower bound is attainable.

MyHill-Nerode Theorem

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

MyHill-Nerode Theorem

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

Constructive proof:

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

Constructive proof:

$K = \{[x] \mid x \in \Sigma^\star\}$ the set of equivalence classes under $\approx_L$

MyHill-Nerode Theorem

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

Constructive proof:

$K = \{[x] \mid x \in \Sigma^\star\}$ the set of equivalence classes under $\approx_L$

$s = [e]$

MyHill-Nerode Theorem

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

Constructive proof:

$K = \{[x] \mid x \in \Sigma^\star\}$ the set of equivalence classes under $\approx_L$

$s = [e]$

$F = \{[x] \mid x \in L\}$

MyHill-Nerode Theorem

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

Constructive proof:

$K = \{[x] \mid x \in \Sigma^\star\}$ the set of equivalence classes under $\approx_L$

$s = [e]$

$F = \{[x] \mid x \in L\}$

$\delta([x], a) = [xa]$ for any $[x] \in K$ and $a \in \Sigma$

MyHill-Nerode Theorem

Theorem (Myhill-Nerode)

Let $L \subseteq \Sigma^\star$ be a regular language. Then there is a deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$ such that $L = L(M)$ and $|K|$ is the number of equivalence classes of $\approx_L$.

Constructive proof:

$K = \{[x] \mid x \in \Sigma^\star\}$ the set of equivalence classes under $\approx_L$

$s = [e]$

$F = \{[x] \mid x \in L\}$

$\delta([x], a) = [xa]$ for any $[x] \in K$ and $a \in \Sigma$

Example

Consider M for $\mathcal{L}((ab \cup ba)^\star)$ and $\mathcal{L}(a^\star)$.

MyHill-Nerode Theorem

Corollary

A language L is regular if and only if $\approx_L$ has finitely many equivalence classes.

MyHill-Nerode Theorem

Corollary

A language L is regular if and only if $\approx_L$ has finitely many equivalence classes.

The corollary can be used to show that a language is not regular (show that it has infinitely many equivalence classes).

MyHill-Nerode Theorem

Corollary

A language L is regular if and only if $\approx_L$ has finitely many equivalence classes.

The corollary can be used to show that a language is not regular (show that it has infinitely many equivalence classes).

Example

Show that $L = \{a^n b^n \mid n \geq 0\}$ is not regular by using MyHill-Nerode theorem.

MyHill-Nerode Theorem

Corollary

A language L is regular if and only if $\approx_L$ has finitely many equivalence classes.

The corollary can be used to show that a language is not regular (show that it has infinitely many equivalence classes).

Example

Show that $L = \{a^n b^n \mid n \geq 0\}$ is not regular by using MyHill-Nerode theorem.

For a given regular language L, the Myhill-Nerode theorem can be used to construct an automaton M recognizing L ($L = L(M)$) with minimal number of states. However, the construction requires writing equivalence classes of $\approx_L$, which is not a trivial task.

State Minimization Algorithm

Given a deterministic automaton $M = (K, \Sigma, \delta, s, F)$, generate an automaton M' with minimal number of states satisfying $L(M) = L(M')$:

State Minimization Algorithm

Given a deterministic automaton $M = (K, \Sigma, \delta, s, F)$, generate an automaton M' with minimal number of states satisfying $L(M) = L(M')$:

- For a deterministic automaton, define $A_M \subseteq K \times \Sigma^\star$ as follows:

$$(q, w) \in A_M \text{ iff } (q, w) \vdash_M^\star (f, e) \text{ for some } f \in F$$

State Minimization Algorithm

Given a deterministic automaton $M = (K, \Sigma, \delta, s, F)$, generate an automaton M' with minimal number of states satisfying $L(M) = L(M')$:

- For a deterministic automaton, define $A_M \subseteq K \times \Sigma^\star$ as follows:

$$(q, w) \in A_M \text{ iff } (q, w) \vdash_M^\star (f, e) \text{ for some } f \in F$$

- Two states $q, p \in K$ are said to be equivalent, $p \equiv q$ if

$$\text{for all } z \in \Sigma^\star : (p, z) \in A_M \text{ iff } (q, z) \in A_M$$

State Minimization Algorithm

Given a deterministic automaton $M = (K, \Sigma, \delta, s, F)$, generate an automaton M' with minimal number of states satisfying $L(M) = L(M')$:

- For a deterministic automaton, define $A_M \subseteq K \times \Sigma^\star$ as follows:

$$(q, w) \in A_M \text{ iff } (q, w) \vdash_M^\star (f, e) \text{ for some } f \in F$$

- Two states $q, p \in K$ are said to be equivalent, $p \equiv q$ if

$$\text{for all } z \in \Sigma^\star : (p, z) \in A_M \text{ iff } (q, z) \in A_M$$

- It two states are equivalent, then their equivalence classes under $\sim_M$ is a subset of the same equivalence class under $\approx_{L(M)}$.

State Minimization Algorithm

Given a deterministic automaton $M = (K, \Sigma, \delta, s, F)$, generate an automaton M' with minimal number of states satisfying $L(M) = L(M')$:

- For a deterministic automaton, define $A_M \subseteq K \times \Sigma^\star$ as follows:

$$(q, w) \in A_M \text{ iff } (q, w) \vdash_M^\star (f, e) \text{ for some } f \in F$$

- Two states $q, p \in K$ are said to be equivalent, $p \equiv q$ if

$$\text{for all } z \in \Sigma^\star : (p, z) \in A_M \text{ iff } (q, z) \in A_M$$

- It two states are equivalent, then their equivalence classes under $\sim_M$ is a subset of the same equivalence class under $\approx_{L(M)}$.
- Thus, if $p \equiv q$, they can be merged to obtain an equivalent automaton with smaller number of states.

State Minimization Algorithm

Iterative algorithm for construction of $\equiv$: $\equiv$ is the limit of the sequence $\equiv_0, \equiv_1, \ldots$.

State Minimization Algorithm

Iterative algorithm for construction of $\equiv$: $\equiv$ is the limit of the sequence $\equiv_0, \equiv_1, \ldots$.

- For $p, q \in K$, $q \equiv_n p$ if $(p, z) \in A_M$ iff $(q, z) \in A_M$ for each $z \in \Sigma^\star$ with $|z| \leq n$.

State Minimization Algorithm

Iterative algorithm for construction of $\equiv$: $\equiv$ is the limit of the sequence $\equiv_0, \equiv_1, \ldots$.

- For $p, q \in K$, $q \equiv_n p$ if $(p, z) \in A_M$ iff $(q, z) \in A_M$ for each $z \in \Sigma^\star$ with $|z| \leq n$.
- Thus $q \equiv_n p$ implies that p and q behaves the same with respect to acceptance of strings for strings up to length n.

State Minimization Algorithm

Iterative algorithm for construction of $\equiv$: $\equiv$ is the limit of the sequence $\equiv_0, \equiv_1, \ldots$.

- For $p, q \in K$, $q \equiv_n p$ if $(p, z) \in A_M$ iff $(q, z) \in A_M$ for each $z \in \Sigma^\star$ with $|z| \leq n$.
- Thus $q \equiv_n p$ implies that p and q behaves the same with respect to acceptance of strings for strings up to length n.
- $q \equiv_0 p$ if p and q are both accepting or both non-accepting (two equivalence classes under $\equiv_0$).

State Minimization Algorithm

Iterative algorithm for construction of $\equiv$: $\equiv$ is the limit of the sequence $\equiv_0, \equiv_1, \ldots$.

- For $p, q \in K$, $q \equiv_n p$ if $(p, z) \in A_M$ iff $(q, z) \in A_M$ for each $z \in \Sigma^\star$ with $|z| \leq n$.
- Thus $q \equiv_n p$ implies that p and q behaves the same with respect to acceptance of strings for strings up to length n.
- $q \equiv_0 p$ if p and q are both accepting or both non-accepting (two equivalence classes under $\equiv_0$).
- $\equiv_{i+1}$ is a refinement of $\equiv_i$. Next lemma relates $\equiv_n$ to $\equiv_{n-1}$.

State Minimization Algorithm

Iterative algorithm for construction of $\equiv$: $\equiv$ is the limit of the sequence $\equiv_0, \equiv_1, \ldots$.

- For $p, q \in K$, $q \equiv_n p$ if $(p, z) \in A_M$ iff $(q, z) \in A_M$ for each $z \in \Sigma^\star$ with $|z| \leq n$.
- Thus $q \equiv_n p$ implies that p and q behaves the same with respect to acceptance of strings for strings up to length n.
- $q \equiv_0 p$ if p and q are both accepting or both non-accepting (two equivalence classes under $\equiv_0$).
- $\equiv_{i+1}$ is a refinement of $\equiv_i$. Next lemma relates $\equiv_n$ to $\equiv_{n-1}$.

Lemma

For p, q and $n \geq 1$, $q \equiv_n p$ if and only if

1. *$q \equiv_{n-1} p$ and*
2. *for all $a \in \Sigma$, $\delta(q, a) \equiv_{n-1} \delta(p, a)$*

State Minimization Algorithm

1. Equivalence classes of $\equiv_0$ are $K \setminus F$ and K
2. Until $\equiv_n$ is the same as $\equiv_{n-1}$
3. compute equivalence class of $\equiv_n$ from $\equiv_{n-1}$ (for each $a \in \Sigma$, and equivalence class $[q]$, compute $\{\delta(q', a) \mid q' \in [q]\}$, and split the equivalence classes that intersect with this set)