CS310 Automata Theory – 2017-2018

Nutan Limaye

Indian Institute of Technology, Bombay
nutan@cse.iitb.ac.in

Module 1: Finite state automata

Credit Structure

Course credit structure

quizzes	30%
mid-sem	30%
end-sem	40%

Office hours:	1 hour per week (Slot: TBA)
Problem solving session:	1 hour per week (Slot: TBA)

Course Outline

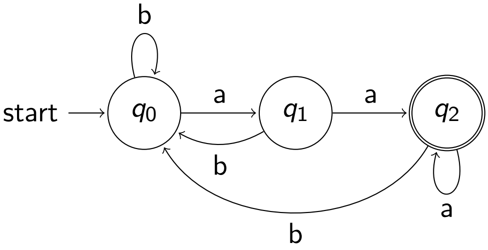
- Regular languages, DFA/NFA, related topics.
- Pushdown automata, context-free languages, other models of computation.
- Turing machines and computability.
- Effective computation, NP vs. P, one-way functions.

Finite state automata

Example

Input: Text file over the alphabet $\{a, b\}$

Check: does the file end with the string 'aa'

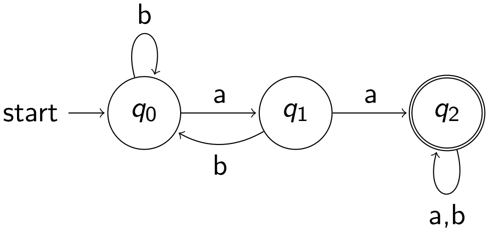


Finite state automata

Example

Input: Text file over the alphabet $\{a, b\}$

Check: does the file contain the string 'aa'

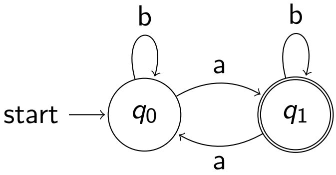


Finite state automata

Example

Input: $w \in \{a, b\}^*$

Check: does w have odd number of as? i.e. is $\#_a(w) \equiv 1 \ (mod 2)$?

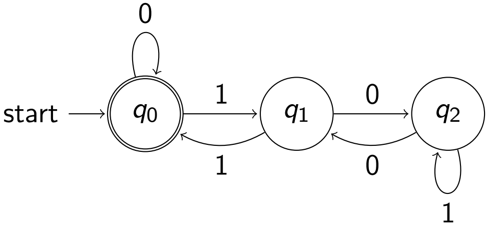


Finite state automata

Example

Input: $w \in \{0, 1\}^*$

Check: is the number represented by w in binary a multiple of 3?



Definition of finite state automata

Definition (DFA)

A deterministic finite state automaton (DFA) $A = (Q, \Sigma, q_0, F, \delta)$, where

- Q is a set of states,
- Σ is the input alphabet,
- q_0 is the initial state,
- $F \subseteq Q$ is the set of final states,
- δ is a set of transitions, i.e. $\delta \subseteq Q \times \Sigma \times Q$ such that $\forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| \leq 1$.

Acceptance by DFA

Definition (Acceptance by DFA)

A deterministic finite state automaton (DFA) $A = (Q, \Sigma, q_0, F, \delta)$, is said to accept a word $w \in \Sigma^*$, where $w = w_1 w_2 \dots w_n$ if

there exists a sequence of states $p_0, p_1, \dots p_n$ s.t.

$p_0 = q_0$,

$p_n \in F$,

$\delta(p_i, w_{i+1}) = p_{i+1}$ for all $0 \leq i \leq n$,

where δ is a set of transitions.

Regular languages

Definition

A language $L \subseteq \Sigma^*$ is a said to be accepted by a DFA A if $L = \{w \mid w \text{ is accepted by } A\}$.

Definition (REG)

A language is said to be a regular language if it is accepted by some DFA.

Examples

$L = \{w \in \{a,b\}^* \mid w \text{ ends with aa}\}$

$L' = \{w \in \{a,b\}^* \mid w \text{ contains aa}\}$

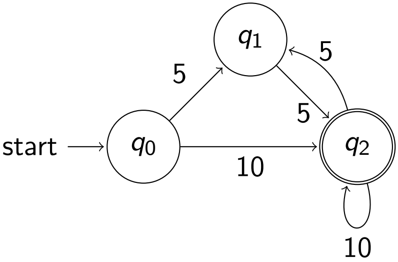
$L_{odd} = \{w \in \{a,b\}^* \mid w \text{ contains odd number of a}\}$

$L_3 = \{w \in \{0,1\}^* \mid w \text{ encodes a number in binary divisible by 3}\}$

Day-to-day examples of finite state automata

Finite state machines are everywhere!

A vending machine that sells objects at Rs. 10 each and can take either Rs. 5 or Rs. 10 coins as input.



Other applications

Finite state machines in many electrinic devices

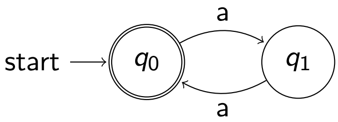
- Automatic coffee dispenser
- Public washing machines
- Fan regulators
- the list can go on!

Closure properties of regular languages

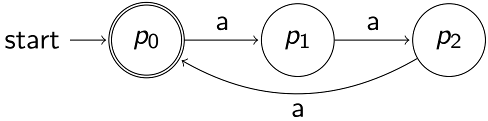
Example

Let $\Sigma = \{a\}$ for this example.

Let $L_1 = \{w \mid |w| \equiv 0 \ (mod\ 2)\}$



Let $L_2 = \{w \mid |w| \equiv 0 \ (mod\ 3)\}$



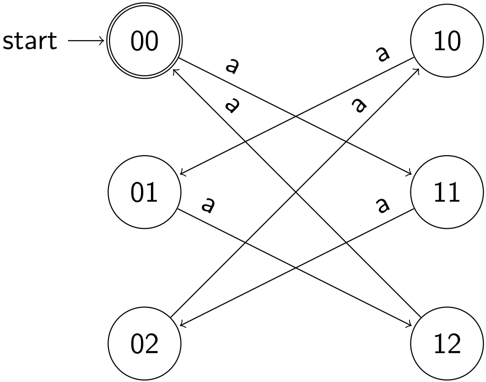
What is $L_1 \cap L_2$?

$L_1 \cap L_2 = \{w \mid |w| \equiv 0 \ (mod\ 6)\}$

Closure properties of regular languages

Example continued

$$L_1 \cap L_2 = \{w \mid |w| \equiv 0 \ (mod\ 6)\}$$



Closure properties of regular languages

Lemma

Let $L_1, L_2 \subseteq \Sigma^*$ be two regular languages, then $L_1 \cap L_2$ is also a regular language.

Proof.

Product construction

Let $A_1 = (Q_1, \Sigma, q_0^1, F_1, \delta_1)$ and $A_2 = (Q_2, \Sigma, q_0^2, F_2, \delta_2)$ be the automata accepting L_1, L_2, respectively.

Let A be a finite state automaton $(Q, \Sigma, q_0, F, \delta)$ s.t.

$$
\begin{array}{ccl}
Q & = & \{(q, q') \mid q \in Q_1, q' \in Q_2\} \\
q_0 & = & (q_0^1, q_0^2) \\
F & = & \{(q, q') \mid q \in F_1, q' \in F_2\} \\
\delta((q, q'), a) & = & (\delta_1(q, a), \delta_2(q', a))
\end{array}
$$

Correctness

$\forall w \in \Sigma^*$, w is accepted by A iff w is accepted by both A_1 and A_2.

Closure properties of regular languages

Lemma

Let $L_1, L_2 \subseteq \Sigma^*$ be two regular languages, then $L_1 \cup L_2$ is also a regular language.

Proof.

Product construction

Let $A_1 = (Q_1, \Sigma, q_0^1, F_1, \delta_1)$ and $A_2 = (Q_2, \Sigma, q_0^2, F_2, \delta_2)$ be the automata accepting L_1, L_2, respectively.

Let A be a finite state automaton $(Q, \Sigma, q_0, F, \delta)$ s.t.

$$\begin{array}{rcl} Q & = & \{(q, q') \mid q \in Q_1, q' \in Q_2\} \\ q_0 & = & (q_0^1, q_0^2) \\ F & = & \{(q, q') \mid q \in F_1 \text{ or } q' \in F_2\} \\ \delta((q, q'), a) & = & (\delta_1(q, a), \delta_2(q', a)) \end{array}$$

Correctness

$\forall w \in \Sigma^*$, w is accepted by A iff w is accepted by either A_1 or A_2.

Closure properties of regular languages

Lemma

Let $L \subseteq \Sigma^*$ be a regular language, then $\overline{L} = \{w \mid w \notin L\}$ is also a regular language.

Proof.

Let $A = (Q, \Sigma, q_0, F, \delta)$ be the automata accepting L.

Let A' be a finite state automaton $(Q', \Sigma', q_0', F', \delta')$ s.t.

$$
\begin{aligned}
Q' &= Q \\
q_0' &= q_0 \\
F' &= \{q \in Q \mid q \notin F\} \\
\delta' &= \delta
\end{aligned}
$$

Correctness

$\forall w \in \Sigma^*$, w is accepted by A' iff w is not accepted by A.

□

Non-deterministic finite state automata

Informal description: A finite state automaton which can branch out to different states on the same letter.

Definition (NFA)

A non-deterministic finite state automaton (NFA) $A = (Q, \Sigma, q_0, F, \delta)$, where

Q is a set of states,

Σ is the input alphabet, also contains empty string, i.e. ϵ ,

q_0 is the initial state,

$F \subseteq Q$ is the set of final states,

δ is a set of transitions, i.e. $\delta \subseteq Q \times \Sigma \times Q$

$\cancel{\forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| \leq 1}$.

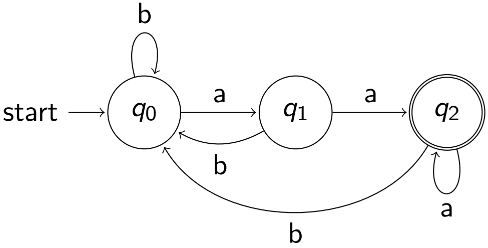
$\forall q \in Q, \forall a \in \Sigma, \delta(q, a) \subseteq Q$.

Non-deterministic finite state automata

Example

Input: Text file over the alphabet $\{a, b\}$

Check: does the file end with the string 'aa'

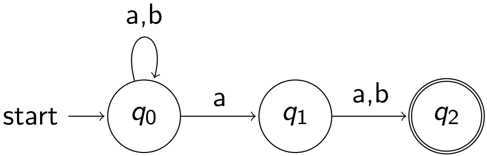


Non-deterministic finite state automata

Example

Input: $w \in \{a, b\}^*$

Check: Is a the second-last letter of w?



Non-deterministic finite state automata

Informal description: A finite state automaton which can branch out to different states on the same letter.

Definition (NFA)

A non-deterministic finite state automaton (NFA) $A = (Q, \Sigma, q_0, F, \delta)$, where

- Q is a set of states,
- Σ is the input alphabet, also contains empty string, i.e. ϵ ,
- q_0 is the initial state,
- $F \subseteq Q$ is the set of final states,
- δ is a set of transitions, i.e. $\delta \subseteq Q \times \Sigma \times Q$
  $\cancel{\forall q \in Q, \forall a \in \Sigma, |\delta(q, a)| \leq 1}$.
  $\forall q \in Q, \forall a \in \Sigma, \delta(q, a) \subseteq Q$.

Acceptance by NFA

Definition (Acceptance by NFA)

A non-deterministic finite state automaton (NFA) $A = (Q, \Sigma, q_0, F, \delta)$, is said to accept a word $w \in (\Sigma \setminus \{\epsilon\})^*$, where $w = w_1 w_2 \ldots w_n$ if

- w can be written as $y_1 y_2 \ldots y_m$, where each $y_i \in \Sigma$ and $m \geq n$
- there exists a sequence of states $p_0, p_1, \ldots p_m$ s.t.
- $p_0 = q_0$,
- $p_m \in F$,
- $p_{i+1} \in \delta(p_i, y_{i+1})$ for all $0 \leq i \leq m - 1$.

An NFA A is said to accept a language L if $L = \{w \mid A \text{ accepts } w\}$.

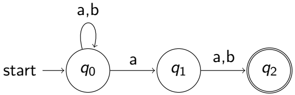
Notation: Let A be an NFA/DFA. We use $L(A)$ to denote the language recognized by A.

Power of NFAs

Lemma

Let A be an NFA. Then $L(A)$ is a regular language. That is, NFA and DFA accept the same set of languages.

We will work it out for an example.



	$\varnothing$	$\{0\}$	$\{1\}$	$\{2\}$	$\{0,1\}$	$\{0,2\}$	$\{1,2\}$	$\{0,1,2\}$
a	$\varnothing$	$\{0,1\}$	$\{2\}$	$\varnothing$	$\{0,1,2\}$	$\{0,1\}$	$\{2\}$	$\{0,1,2\}$
b	$\varnothing$	$\{0\}$	$\{2\}$	$\varnothing$	$\{0,2\}$	$\{0\}$	$\{2\}$	$\{0,2\}$

Subset construction

Lemma

Let A be an NFA. Then $L(A)$ is a regular language. That is, NFA and DFA accept the same set of languages.

Proof.

Let $A = (Q, \Sigma, q_0, F, \delta)$. We will construct a DFA $B = (Q', \Sigma, q_0', F', \Delta)$ such that $L(A) = L(B)$.

Subset construction

$$Q' = 2^Q,$$

$$q_0' = \{q_0\},$$

$$F' = \{S \subseteq Q \mid S \cap F \neq \varnothing\}.$$

$$\Delta(S, a) = \textstyle\bigcup_{p \in S} \delta(p, a).$$

$\square$

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

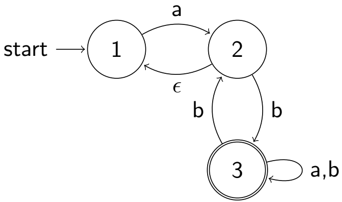
Proof Idea

Let $S \subseteq Q$.

Let

$$E(S) = \left\{ q \;\middle|\; \begin{array}{l} q \text{ is reachable from some state in } S \\ \text{with zero or more } \epsilon \text{ transitions} \end{array} \right\}$$

Example



$$\begin{array}{rcl} E(\{1\}) & = & \{1\} \\ E(\{2\}) & = & \{1,2\} \\ E(\{3\}) & = & \{3\} \end{array}$$

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

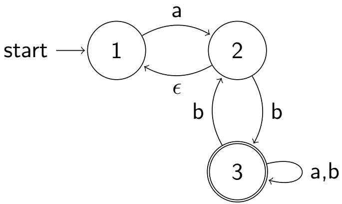
Proof Idea

Let $S \subseteq Q$.

Let

$$E(S) = \left\{ q \;\middle|\; \begin{array}{l} q \text{ is reachable from some state in } S \\ \text{with zero or more } \epsilon \text{ transitions} \end{array} \right\}$$

Example



$$\begin{aligned} \delta'(1, a) &= E(\delta(1, a)) \\ &= E(\{2\}) \\ &= \{1, 2\} \end{aligned}$$

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

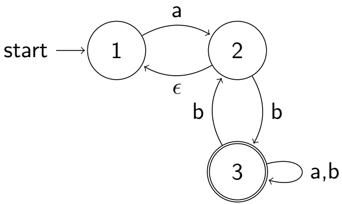
Proof Idea

Let $S \subseteq Q$.

Let

$$E(S) = \left\{ q \;\middle|\; \begin{array}{l} q \text{ is reachable from some state in } S \\ \text{with zero or more } \epsilon \text{ transitions} \end{array} \right\}$$

Example



$$\begin{aligned} \delta'(1, a) &= E(\delta(1, a)) \\ &= E(\{2\}) \\ &= \{1, 2\} \end{aligned}$$

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

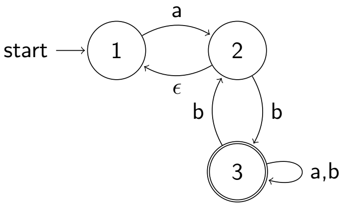
Proof Idea

Let $S \subseteq Q$.

Let

$$E(S) = \left\{ q \;\middle|\; \begin{array}{l} q \text{ is reachable from some state in } S \\ \text{with zero or more } \epsilon \text{ transitions} \end{array} \right\}$$

Example



$$\begin{aligned} \delta'(3,b) &= E(\delta(3,b)) \\ &= E(\{2,3\}) \\ &= \{1,2,3\} \end{aligned}$$

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

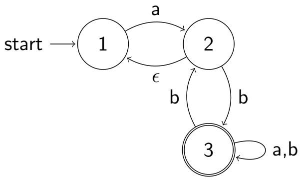
Proof Idea

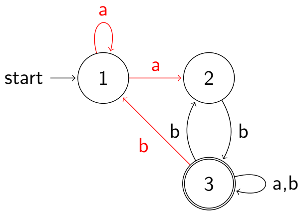
Let $S \subseteq Q$.

Let

$$E(S) = \left\{ q \;\middle|\; \begin{array}{l} q \text{ is reachable from some state in } S \\ \text{with zero or more } \epsilon \text{ transitions} \end{array} \right\}$$

Example





Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

Proof.

Let $A = (Q, \Sigma, q_0, F, \delta)$ be an NFA with ϵ transitions. We construct NFA, say B as follows:

Construction

$Q' = Q$,

Σ' same as Σ, but no ϵ used anywhere,

$\delta'(q, a) = E(\delta(q, a))$,

$q_0' = q_0$,

$F' = F$.

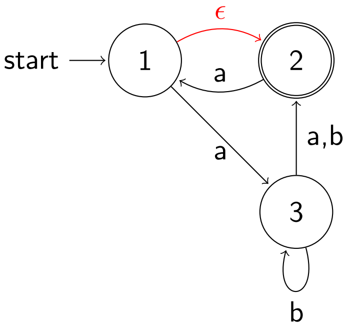
There can be ϵ transitions from the start state or to the final state. □

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

Example

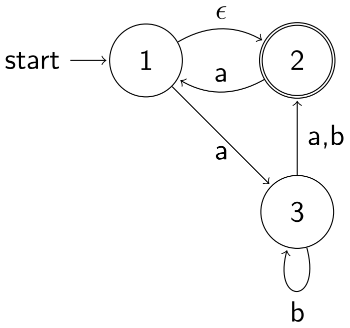


Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

Example



Add a new start state $\tilde{q}_0$.

Consider $\delta(p, c)$ for every $p \in E(q_0)$ and $c \in \Sigma$.

Add an edge from $\tilde{q}_0$ to $q \in Q$ with label c if

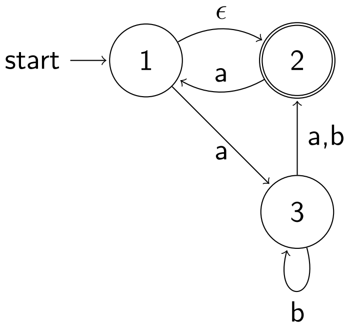
$$q \in E\left(\bigcup_{p \in E(q_0)} \delta(p, c)\right).$$

Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

Example



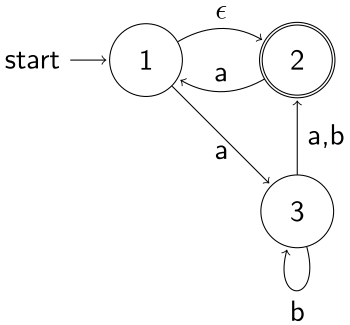
$$
\begin{aligned}
\text{As } & E(\{1\}) = \{1,2\} \\
& E(\cup_{p\in\{1,2\}}\delta(p,a)) \\
= \quad & E(\delta(1,a) \cup \delta(2,a)) \\
= \quad & E(\{3,1\}) \\
= \quad & \{1,2,3\}
\end{aligned}
$$

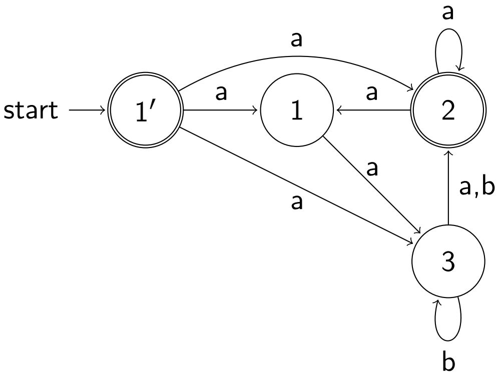
Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

Example





Hanlding the ϵ moves

Lemma

For any NFA A with ϵ transitions, there is another NFA, say B, such that B has no ϵ transitions and $L(A) = L(B)$.

Proof.

Let $A = (Q, \Sigma, q_0, F, \delta)$ be given. We construct $B = (Q', \Sigma', q_0, F', \delta')$ as follows:

Construction

$$Q' = Q \cup \{\tilde{q}_0\},\ q'_0 = \tilde{q}_0,\ \Sigma' \text{ same as } \Sigma \text{ but no } \epsilon,$$

$$F' = \begin{cases} F \cup \{\tilde{q}_0\} & \text{if } E(\{q_0\}) \cap F \neq \emptyset \\ F & \text{otherwise} \end{cases}$$

$$\delta'(q, a) = \begin{cases} E(\delta(E(q_0), a)) & \text{if } q = \tilde{q}_0 \\ E(\delta(q, a)) & \text{otherwise} \end{cases}$$

□

Regular expressions

Various expressions formed by $+, \circ, *$ *operators on* Σ.

Definition (Regular expression)

The following are regular expressions:

1. ϵ,
2. $a, \forall a \in \Sigma$,
3. $\varnothing$,
4. $R_1 + R_2$,
5. $R_1 \circ R_2$,
6. R_1^*,

where, R_1, R_2 are regular expressions.

Example

$$\Sigma^* a \Sigma^* = \{w \mid w \text{ contains at least one } a\}$$

$$(\Sigma\Sigma)^* = w \mid |w| \equiv 0(mod2)$$

Language defined by a regular expression

Definition (Language defined by regular expression)

The language defined by a regular expression is:

1. $L(\epsilon) = \epsilon$,
2. $L(a) = \{a\}, \forall a \in \Sigma$,
3. $L(\varnothing) = \varnothing$,
4. $L(R_1 + R_2) = L(R_1) \cup L(R_2)$
5. $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$,
6. $L(R_1^*) = (L(R_1))^*$,

where, R_1, R_2 are regular expressions.

Lemma

The language defined by any regular expression is regular.

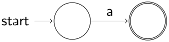
Language defined by regular expression

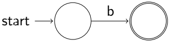
Lemma

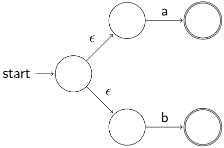
The language defined by any regular expression is regular.

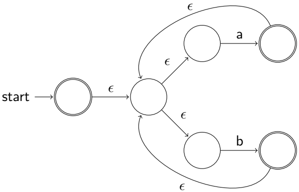
Example

$$(a+b)^*$$









Language defined by regular expression

Lemma

The language defined by any regular expression is regular.

Proof idea

It is easy to construct NFAs for 1.,2.,3.

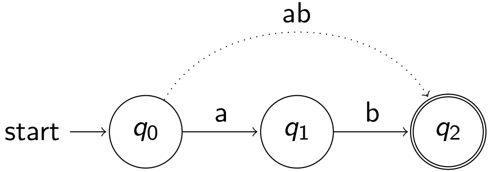
If we inductively have NFAs for $L(R_1), L(R_2)$ then we can create an NFA for $L(R_1 + R_2)$ and $L(R_1 \circ R_2)$.

Similarly, if we inductively have NFAs for $L(R_1)$ then we can create an NFA for $(L(R_1))^*$

DFA to regular expression

Transitive closure method

Example



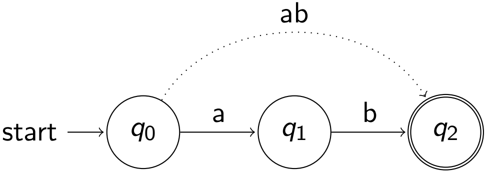
In general compute $R_{i,j}$, the regular expression arising while going from state i to state j.

Construct $R_{i,j}$ for every pair of state i, j.

DFA to regular expression

Transitive closure method

Example



DFA to regular expression

Transitive closure method: an exercise in dynamic programming

Assume there is some ordering on the states of the automaton.

Let $R_{i,j}^{k}$ denote the set of all strings that take the automaton from q_i to q_j without passing through a state numbered larger than q_k.

We can build $R_{i,j}^{1}, R_{i,j}^{2}, \ldots, R_{i,j}^{|Q|}$ recursively as follows:

$$R_{i,j}^{k} = R_{i,j}^{k-1} + R_{i,k}^{k-1} \cdot (R_{k,k}^{k-1})^{*} \cdot R_{k,j}^{k-1}.$$

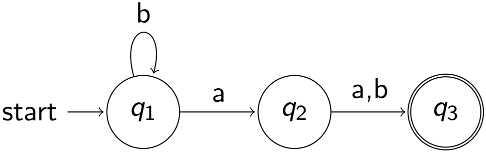
We also need to initialize $R_{i,j}^{0}$ for all pairs i, j as follows:

$$R_{i,j}^{0} = \begin{cases} a & \text{if } i \neq j \text{ and } \delta(q_i, a) = q_j \\ a + \epsilon & \text{if } i = j \text{ and } \delta(q_i, a) = q_j \\ \epsilon & \text{if } i = j \text{ and } \delta(q_i, a) \neq q_j \\ \varnothing & \text{otherwise.} \end{cases}$$

DFA to regular expression

Transitive closure method:

Example



$$R^0_{1,1} = b + \epsilon,\ R^0_{1,2} = a,\ R^0_{2,2} = \epsilon,\ R^0_{3,3} = \epsilon.$$
$$R^0_{2,3} = a + b,\ R^0_{1,3},\ R^0_{2,1},\ R^0_{3,1},\ R^0_{3,2} = \emptyset.$$

$$R^1_{1,1} = (b+\epsilon) + (b+\epsilon)(b+\epsilon)^*(b+\epsilon) = b^*$$
$$R^1_{1,2} = a + (b+\epsilon)(b+\epsilon)^* a = b^* a$$
$$R^1_{2,2} = \epsilon,\ R^1_{3,3} = \epsilon$$
$$R^1_{2,3} = a + b,\ R^1_{1,3},\ R^1_{2,1},\ R^1_{3,1},\ R^1_{3,2} = \emptyset.$$

$$R^2_{1,3} = \emptyset + b^* a(\epsilon)^*(a+b) = b^* a(a+b).$$

Proving that PAL is not a regular language

Lemma

$\forall n \in \mathbb{N}$ let $PAL_n = \{w \cdot w^R \mid w \in \Sigma^*, |w| = n\}$. Any automaton accepting PAL_n must have $|\Sigma|^n$ states.

Proof.

By Pigeon Hole Principle.

Suppose $\exists x, y \in \Sigma^n$ such that $x \neq y$,

automaton reaches the same state after reading both x, y.

Then $x \cdot x^R$ and $y \cdot x^R$ are both accepted or both rejected,

which is a contradiction.

□

Corollary

Let $PAL = \cup_{n \geq 0} PAL_n$. PAL is not regular.

Proving that $L_{a,b}$ is not a regular language

Lemma

There is no finite state automaton accepting $L_{a,b}$, where $L_{a,b} = \{a^n b^n \mid n \geq 0\}$.

Proof.

By Pigeon Hole Principle.

Suppose $\exists i, j \in \mathbb{N}$ such that $i \neq j$,
automaton reaches the same state after reading both a^i, a^j.

Then $a^i \cdot b^j$ and $a^j \cdot b^j$ are both accepted or both rejected,
which is a contradiction.

$\square$

Pumping lemma

A recipe for proving that a given language is non-regular.

Lemma (Pumping Lemma)

If L is a regular language, then $\exists p \in \mathbb{N}$ such that for any strings x, y, z with $x \cdot y \cdot z \in L$ and $|y| \geq p$,

1. *there exist strings u, v, w, s.t. y can be written as $y = u \cdot v \cdot w$,*
2. $\forall i \geq 0$ $x \cdot u \cdot v^i \cdot w \cdot z \in L$,
3. $|v| > 0$.

To prove that a given language L is not regular, the contrapositive of the above statement is useful.

Contrapositive of the pumping lemma

Lemma

We say that a language L satisfies **Property-NR** *if the following conditions hold:*

- ☹ $\forall p \geq 0$,
- ☺ $\exists x, y, z$ *such that* $x \cdot y \cdot z \in L$ *and* $|y| \geq p$,
- ☹ $\forall u, v, y$ *such that* $|v| > 0$, $y = u \cdot v \cdot w$,
- ☺ $\exists i \; x \cdot u \cdot v^i \cdot w \cdot z \notin L$.

If L satisfies Property-NR then L is not regular.

Using the pumping lemma

We say that a language L satisfies **Property-NR** if the following conditions hold:

- ☹ $\forall p \geq 0$,
- ☺ $\exists x, y, z$ such that $x \cdot y \cdot z \in L$ and $|y| \geq p$,
- ☹ $\forall u, v, y$ such that $|v| > 0$, $y = u \cdot v \cdot w$,
- ☺ $\exists i \; x \cdot u \cdot v^i \cdot w \cdot z \notin L$.

If L satisfies Property-NR then L is not regular.

We will now use the lemma to prove that $L_{a,b} = \{a^n b^n \mid n \geq n\}$ is not regular.

For any chosen $p \geq 0$, let $x := a^p$, $y := b^p$, $z = \epsilon$.

For any split of y as $u \cdot v \cdot w$, if we take $x \cdot u \cdot v^i \cdot w = 0^p 1^q$, where $q > p$ as long as $i > 0$.

In particular, $x \cdot u \cdot v^2 \cdot w \cdot z \notin L$.

Applications of pumping lemma

Let $L = \{ww^R \mid w \in \Sigma^*\}$

- ☹ For any chosen p,
- ☺ let $x = \epsilon$, $y = 0^p$, $z = 110^p$.
- ☹ For any split of y into u, v, w
- ☺ $xuv^iwz = 0^q110^p$, as long as $i > 0$.
  In particular, $xuv^2wz \notin L$.

We say that a language L satisfies **Property-NR**
if the following conditions hold:

- ☹ $\forall p \geq 0$,
- ☺ $\exists x, y, z$ such that $x \cdot y \cdot z \in L$ and $|y| \geq p$,
- ☹ $\forall u, v, y$ such that $|v| > 0$, $y = u \cdot v \cdot w$,
- ☺ $\exists i\ x \cdot u \cdot v^i \cdot w \cdot z \notin L$.

If L satisfies Property-NR then L is not regular.

Applications of pumping lemma

$L = \{a^q \mid q \text{ is a prime number }\}$

- ☹ For any chosen p,
- ☺ let $x, z = \epsilon$, $y = a^n$, $n \geq p$ and a prime.
- ☹ For any split of y into u, v, w
- ☺ $xuv^{n+1}wz = a^{n(k+1)}$, where $k := |v|$.
  That is, $xuv^{n+1}wz = a^{n(k+1)} \notin L$.

We say that a language L satisfies **Property-NR**
if the following conditions hold:

- ☹ $\forall p \geq 0$,
- ☺ $\exists x, y, z$ such that $x \cdot y \cdot z \in L$ and $|y| \geq p$,
- ☹ $\forall u, v, y$ such that $|v| > 0$, $y = u \cdot v \cdot w$,
- ☺ $\exists i \; x \cdot u \cdot v^i \cdot w \cdot z \notin L$.

If L satisfies Property-NR then L is not regular.

Building on pumping lemma

The following language is not regular:

$$EQ = \{w \in \{a,b\}^* \mid \#_a(w) = \#_b(w)\}$$

Suppose D is regular.

$D \cap L(a^* b^*)$ is also regular, as the intersection of two regular languages is regular and any regular expression defines a regular language.

But $D \cap L(a^* b^*) = \{a^n b^n \mid n \geq 0\}$ is not regular, which we proved using the pumping lemma.

Pumping down

Let $L = \{0^i 1^j \mid i, j \in \mathbb{N} \text{ and } i > j\}$.

For any choice of $p \geq 0$,

Let $x = \epsilon$, $y = 0^{p+1}$, $z = 1^p$.

Then $x \cdot y \cdot z \in L$.

Now for any choice of u, v, w, s.t $u \cdot v \cdot w = y$ and $|v| > 0$

$x \cdot u \cdot v^0 \cdot w \cdot z = 0^{p'} 1^p$, where $p' \leq p$.

$\therefore x \cdot u \cdot v^0 \cdot w \cdot z \notin L$.

Relations on Σ

Let R be an equivalence relation on the set Σ^*, i.e. $R \subseteq \Sigma^* \times \Sigma^*$ such that

REFLEXIVE $\forall x \in \Sigma^* \; R(x, x)$ holds.

SYMMETRIC $\forall x, y \in \Sigma^* \; R(x, y) = R(y, x)$ hold.

TRANSITIVE $\forall x, y, z \in \Sigma^*$ if $R(x, y), R(y, z)$ hold then $R(x, z)$ also holds.

Transition function δ extended to δ^*

Recall the definition from Tutorial 2

Definition

Given a DFA $A = (Q, \Sigma, q_0, F, \delta)$, let $\delta^* : Q \times \Sigma^* \to Q$ be the function defined inductively as follows:

for any $q \in Q$, $\delta^*(q, \epsilon) = q$

for any $q \in Q, w \in \Sigma^*$ and $a \in \Sigma$, $\delta^*(q, wa) = \delta(\delta^*(q, w), a))$.

That is, given a state and a word $w \in \Sigma^*$, δ^* outputs the state in which A ends up, after reading the string w.

Relation of Σ^*

Let L be a regular language recognized by a DFA $A = (Q, \Sigma, \delta, q_0, F)$.

We say that $\forall x, y \in \Sigma^*$

$$x \equiv_A y \quad \text{iff} \quad \underbrace{\delta^*(q_0, x)}_{\substack{\text{state}\\ \text{reached}\\ \text{on } x\\ \text{from } q_0}} \quad = \quad \underbrace{\delta^*(q_0, y)}_{\substack{\text{state}\\ \text{reached}\\ \text{on } y\\ \text{from } q_0}}$$

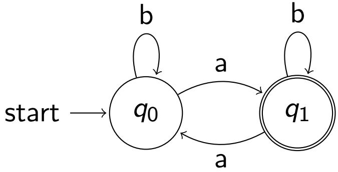
Assume that the auomaton is complete.

Observe that $\equiv_A$ is an equivalence relation.

Example

Example of an equivalence relation.

Consider the following automaton, say A.



$aab \equiv_A abababa$.

$aabaaa \equiv_A a$.

The words with even number of a's form one equivalence class.

The words with odd number of a's form the other equivalence class.

There are no other equivalence classes.

Properties of equivalence relation on Σ^*

Definition (right congruence)

An equivalence relation $\equiv$ defined on Σ^* is said to be **a right congruence** if $\forall x, y \in \Sigma^*$ and $\forall a \in \Sigma$, $x \equiv y \implies x \cdot a \equiv y \cdot a$.

Definition (Refinement)

An equivalence relation $\equiv$ is said to **refine** a language L, if $x \equiv y$ then $(x \in L \Longleftrightarrow y \in L)$.

Definition (Finite index)

An equivalence relation is said to have **finite index** if the number of equivalence classes defined by $\equiv$ is finite.

Lemma

For a DFA A, the equivalence relation $\equiv_A$ defined as before is

is a right congruence, refines $L(A)$, has finite index.

Properties of $\equiv_A$

Lemma

For a DFA A, the equivalence relation $\equiv_A$ defined as before is

is a right congruence, refines $L(A)$, has finite index.

Proof.

right congruence

$$\begin{aligned}\delta^*(q_0, x\cdot a) &= \delta(\delta^*(q_0,x),a)\\ &= \delta(\delta^*(q_0,y),a) \because x \equiv_A y\\ &= \delta^*(q_0, y\cdot a)\end{aligned}$$

finite index

For $q \in Q$,

$[q] := \{w \in \Sigma^* \mid \delta^*(q_0,w) = q\}$

$\#$ equivalence classes $= |Q|$.

refinement

If $x \equiv_A y$

then $\delta^*(q_0,x) = \delta^*(q_0,y)$

$\therefore x, y$ both accepted or

both rejected.

$\square$

Myhill-Nerode relation

Definition

An equivalence relation $\equiv$ on Σ^* is said to be **a Myhill-Nerode relation** for a language L if

- it is a right congruence
- refining L
- and has a finite index.

Lemma (Regular language $\implies$ Myhill-Nerode relation)

For any regular language there is a Myhill-Nerode relation.

What about the converse?

Generalised right contruence

Definition (generalised right congruence)

An equivalence relation $\equiv$ defined on Σ^* is said to be **a generalised right congruence** if $\forall x, y \in \Sigma^*$ and $\forall z \in \Sigma^*$, $x \equiv y \implies x \cdot z \equiv y \cdot z$.

Lemma (right congruence $\Rightarrow$ generalised right congruence)

Let $\equiv$ be an equivalence relation defined on Σ^*. If $\equiv$ is a right congruence then it is also a generalised right congruence.

The proof is by induction. (Problem 3, Tutorial 4.)

From now on we will use generalised right congruence and right congruence interchangeably and call both right congruence.

Non-regular languages

Let $L_{a,b} = \{a^n b^n \mid n \geq 0\}$.

Consider any relation $\equiv$ on $\{a, b\}^*$.

Assume that it is a right congruence and refines L.

Now we will show that it does not have finite index.

For $n \neq m$, say $a^n \equiv a^m$.
By right congruence $a^n \cdot b^n \equiv a^m \cdot b^n$.
But $a^n b^n \in L$ and $a^m b^n \notin L$.

Let $FACTORIAL = \{a^{n!} \mid n \geq 0\}$.

Consider any relation $\equiv$ on $\{a\}^*$.

Assume that it is a right congruence and refines L.

Now we will show that it does not have finite index.

Say $a^{n!} \equiv a^{n+1!}$?
By right congruence $a^{n!} \cdot a^{n \cdot n!} \equiv a^{n+1!} \cdot a^{n \cdot n!}$.
But $a^{n!} \cdot a^{n \cdot n!} \in L$ and $a^{n+1!} \cdot a^{n \cdot n!} \notin L$.

Converse also holds

Lemma

Let $L \subseteq \Sigma^*$. If there is a Myhill-Nerode relation for L then L is regular.

Proof idea

Using the relation, construct a finite state automaton.

Let each equivalence class of the relation be a state of the automaton.

Define transitions naturally.

Converse also holds

Lemma

Let $L \subseteq \Sigma^*$. If there is a Myhill-Nerode relation for L then L is regular.

Proof.

Construction

Let $\equiv$ be a Myhill-Nerode relation.

Let $[x] = \{y \mid y \equiv x\}$.

Let $A_{\equiv} = (Q, \Sigma, \delta, q_0, F)$ be defined as follows:

$Q = \{[x] \mid x \in \Sigma^*\}$,

$q_0 = [\epsilon]$, $F = \{[x] \mid x \in L\}$, $\delta([x], a) = [xa]$.

Correctness: Can be proved using induction. $\square$

Theorem (Myhill-Nerode theorem)

Let $L \subseteq \Sigma^*$. There is a Myhill-Nerode relation for L if and only if L is regular.

Application of Myhill-Nerode theorem

Show that $PAL = \{w \cdot w^R \mid w \in \Sigma^*\}$ is not regular.

> Consider any relation $\equiv$ on $\{a, b\}^*$.
>
> Assume that it is a right congruence and refines PAL.
>
> Now we will show that it does not have finite index.
>
> > For $x \neq y$, say $x \equiv y$.
> > By right congruence $x \cdot x^R \equiv y \cdot x^R$.
> > But $x \cdot x^R \in L$ and $y \cdot x^R \notin L$.
> > Therefore, no two $x \neq y$ are equivalent. Hence $\equiv$ not finite index.

Let $PRIME = \{a^q \mid q \text{ is a prime number}\}$.

> Consider any relation $\equiv$ on $\{a\}^*$.
>
> Assume that it is a right congruence and refines L.
>
> Now show that it does not have finite index.

Decision problems on regular languages

Acceptance problem (for fixed Σ)

Given: DFA A, input string $w \in \Sigma^*$

Output: "yes" iff A accepts w.

Construct a graph from an automaton:

> Let $Q = \{q_0, \ldots, q_{m-1}\}$, q_0 be the start state,
> $F \subseteq Q$ be the set of final states.
>
> Create a layered graph $G_{A,n}$, where $|w| = n$, as follows:
>
> Make $n+1$ copies of Q: $Q_0, Q_1, \ldots, Q_n$, where $Q_i = \{q_{i,0}, \ldots, q_{i,m-1}\}$.
>
> Add edge $(q_{i,u}, q_{i+1,v})$ with label $a \in \Sigma$
> if $\delta(q_u, a) = q_v$.

Lemma

There is a path from $q_{0,0}$ to $q_{n,u}$ labelled by a string w in $G_{A,|w|}$ if and only if $\delta^*(q_0, w) = q_u$ in A.

Decision problems on regular languages

Nonemptiness problem (for fixed Σ)

Given: DFA A

Output: "yes" iff $\exists w : A$ accepts w.

Lemma

If a DFA $A = (Q, \Sigma, \delta, q_0, F)$ accepts some string then it accepts a string of length $\leq |Q|$.

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: DFA B s.t. $L(A) = L(B)$ and B has the smallest number of states possible for recognizing $L(A)$

Definition

Let $A = (Q, \Sigma, q_0, F, \delta)$. We call states p, q indistinguishable if $\forall w \in \Sigma^*,\ \delta^*(p, w) \Leftrightarrow \delta^*(q, w)$.

Definition

Let $A = (Q, \Sigma, \delta, q_0, F)$. We call states p, q equivalent if $\forall w \in \Sigma^*,\ \delta^*(p, w) \in F \Leftrightarrow \delta^*(q, w) \in F$.

Minimization algorithm.

Identify equivalent states.

Collapse them.

Finding equivalent states

Finding equivalent states

Given: DFA A

Output: sets of states of A equivalent to each other

Example

	0	1	2	3	4	5
a	1	2	3	4	5	0

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: sets of states of A equivalent to each other

Example

	0	1	2	3	4	5
a	1	2	3	4	5	0

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

0					
✓	1				
–	✓	2			
–	✓	–	3		
✓	–	✓	✓	4	
–	✓	–	–	✓	5

Minimization problem

Minimization problem (for fixed Σ)

Given:	DFA A
Output:	sets of states of A equivalent to each other

Example

	0	1	2	3	4	5
a	1	2	3	4	5	0

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

0					
✓	1				
✓	✓	2			
–	✓	–	3		
✓	–	✓	✓	4	
–	✓	–	–	✓	5

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: sets of states of A equivalent to each other

Example

	0	1	2	3	4	5
a	1	2	3	4	5	0

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

0					
✓	1				
✓	✓	2			
–	✓	–	3		
✓	–	✓	✓	4	
✓	✓	–	–	✓	5

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: sets of states of A equivalent to each other

Example

	0	1	2	3	4	5
a	1	2	3	4	5	0

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

0					
✓	1				
✓	✓	2			
–	✓	✓	3		
✓	–	✓	✓	4	
✓	✓	–	–	✓	5

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: sets of states of A equivalent to each other

Example

	0	1	2	3	4	5
a	1	2	3	4	5	0

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

0					
✓	1				
✓	✓	2			
–	✓	✓	3		
✓	–	✓	✓	4	
✓	✓	–	✓	✓	5

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: sets of states of A equivalent to each other

Algorithm

Let $Q = \{q_1, \dots, q_n\}$.

1. For each $1 \leq i < j \leq n$, initialize $T(i,j) = --$
2. For each $1 \leq i < j \leq n$

   If $(q_i \in F$ AND $q_j \notin F)$ OR $(q_i \in F$ AND $q_j \notin F)$
   $T(i,j) \leftarrow \checkmark$
3. Repeat

   $\{$ For each $1 \leq i < j \leq n$
   If $\exists a \in \Sigma, T(\delta(q_i, a), \delta(q_j, a)) = \checkmark$
   then $T(i,j) \leftarrow \checkmark$
   $\}$

   Untill T stays unchanged.

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: DFA B s.t. $L(A) = L(B)$ and B has the smallest number of states possible for recognizing $L(A)$

Example

	0	1	2	3	4	5
a	1	3	4	5	5	5
b	2	4	3	5	5	5

(Red color indicates final states.)

Minimization problem

Minimization problem (for fixed Σ)

Given: DFA A

Output: DFA B s.t. $L(A) = L(B)$ and B has the smallest number of states possible for recognizing $L(A)$

Example

	0	1	2	3	4	5
a	1	3	4	5	5	5
b	2	4	3	5	5	5

(Red color indicates final states.)

0					
–	1				
–	–	2			
–	–	–	3		
–	–	–	–	4	
–	–	–	–	–	5

DIY!

Recap of Module - I

DFA, NFA, Regular expressions and their equivalence.

Closure properties of regular languages.

Non-regular languages and Pigeon Hole Principle.

Pumping lemma and its applications.

Myhill Nerode relation and characterization of regular languages.

Polynomial time algorithms for membership problem, emptiness problem and minimization problem.

Module - II: Different models of computation

What do we plan to do in this module?

2DFA, a variant of a DFA where the input head moves right/left.

Chapter 18, from the text of Dexter Kozen

Pushdown automata, context-free languages(CFLs), context-free grammar(CFG), closure properties of CFLs.

Module - II: Different models of computation

2DFA: Two-way deterministic finite state automata.

$$\boxed{\# \;\; w_1 \;\; w_2 \;\; \ldots \;\; \ldots \;\; \ldots \;\; \ldots \;\; \ldots \;\; \ldots \;\; w_n \;\; \$}$$

Input head moves left/right on this tape.

It does not go to the left of $\#$.

It does not go to the right of \$.

Can potentially get stuck in an infinite loop!

Formal definition of 2DFA

Definition

A 2DFA $A = (Q, \Sigma \cup \{\#, \$\}, \delta, q_0, q_{acc}, q_{rej})$, where

Q :	set of states,	Σ:	input alphabet
$\#$:	left endmarker	$\$$:	right endmarker
q_0:	start state		
q_{acc}:	accept state	q_{rej}:	reject state

$$\delta : Q \times (\Sigma \cup \{\#, \$\} \to Q \times \{L, R\}$$

The following conditions are forced:

$\forall q \in Q$, $\exists q', q'' \in Q$ s.t. $\delta(q, \#) = (q', R)$ and $\delta(q, \$) = (q'', L)$.