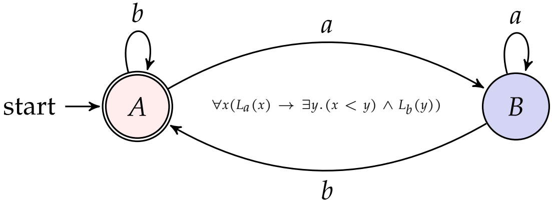
CS 208: Automata Theory and Logic

DFA Equivalence and Minimization

Ashutosh Trivedi



Department of Computer Science and Engineering,
Indian Institute of Technology Bombay.

DFA Equivalence and Minimization

1. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
2. Recall the definition of extended transition function $\hat{\delta}$.
3. Let $L(A, q)$ be the languages $\{w \,:\, \hat{\delta}(q, w) \in F\}$
4. Recall the language of A is defined as $L(A) = L(A, q_0)$.
5. Two states $q_1, q_2 \in Q$ are equivalent if $L(A, q_1) = L(A, q_2)$.
6. We say that two DFAs A_1 and A_2 are equivalent iff $L(A_1) = L(A_2)$.

Theorem (DFA Equivalence)

For every DFA there exists a unique (up to state naming) minimal DFA.

DFA Equivalence and Minimization

1. Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA.
2. Recall the definition of extended transition function $\hat{\delta}$.
3. Let $L(A, q)$ be the languages $\{w \,:\, \hat{\delta}(q, w) \in F\}$
4. Recall the language of A is defined as $L(A) = L(A, q_0)$.
5. Two states $q_1, q_2 \in Q$ are equivalent if $L(A, q_1) = L(A, q_2)$.
6. We say that two DFAs A_1 and A_2 are equivalent iff $L(A_1) = L(A_2)$.

Theorem (DFA Equivalence)

For every DFA there exists a unique (up to state naming) minimal DFA.

How to minimize DFAs?

How to minimize a DFA?

Two observations:

– Removing unreachable states: removing states unreachable from the start state does not change the language accepted by a DFA.

How to minimize a DFA?

Two observations:

- Removing unreachable states: removing states unreachable from the start state does not change the language accepted by a DFA.
- Merging equivalent states: merging equivalent states does not change the language accepted by a DFA.

How to minimize a DFA?

Two observations:

- Removing unreachable states: removing states unreachable from the start state does not change the language accepted by a DFA.
- Merging equivalent states: merging equivalent states does not change the language accepted by a DFA.

Algorithms:

1. Breadth-first search or depth-first search (to identify reachable states)
2. table-filling algorithm (by E. F. Moore) (other algorithms exist due to Hopcroft and Brzozowski)

Table-filling algorithm

- Two states are distinguishable if they are not equivalent.
- Formally, two states q_1, q_2 are distinguishable, if there exists a string $w \in \Sigma^*$ such that exactly one of $\hat{\delta}(q_1, w)$ and $\delta(q_2, w)$ is an accepting state.
- Table-filling algorithm is recursive discovery of distinguishable pairs.

Table-filling algorithm

- Two states are distinguishable if they are not equivalent.
- Formally, two states q_1, q_2 are distinguishable, if there exists a string $w \in \Sigma^*$ such that exactly one of $\hat{\delta}(q_1, w)$ and $\delta(q_2, w)$ is an accepting state.
- Table-filling algorithm is recursive discovery of distinguishable pairs.
- Basis: Pair (p, q) is distinguishable if $p \in F$ and $q \notin F$.

Table-filling algorithm

- Two states are distinguishable if they are not equivalent.
- Formally, two states q_1, q_2 are distinguishable, if there exists a string $w \in \Sigma^*$ such that exactly one of $\hat{\delta}(q_1, w)$ and $\delta(q_2, w)$ is an accepting state.
- Table-filling algorithm is recursive discovery of distinguishable pairs.
- Basis: Pair (p, q) is distinguishable if $p \in F$ and $q \notin F$. why?

Table-filling algorithm

- Two states are distinguishable if they are not equivalent.
- Formally, two states q_1, q_2 are distinguishable, if there exists a string $w \in \Sigma^*$ such that exactly one of $\hat{\delta}(q_1, w)$ and $\delta(q_2, w)$ is an accepting state.
- Table-filling algorithm is recursive discovery of distinguishable pairs.
- Basis: Pair (p, q) is distinguishable if $p \in F$ and $q \notin F$. why?
- Induction: Pair (p, q) is distinguishable if states $\delta(p, a)$ and $\delta(q, a)$ are distinguishable for some $a \in \Sigma$.

Table-filling algorithm

- Two states are distinguishable if they are not equivalent.
- Formally, two states q_1, q_2 are distinguishable, if there exists a string $w \in \Sigma^*$ such that exactly one of $\hat{\delta}(q_1, w)$ and $\delta(q_2, w)$ is an accepting state.
- Table-filling algorithm is recursive discovery of distinguishable pairs.
- Basis: Pair (p, q) is distinguishable if $p \in F$ and $q \notin F$. why?
- Induction: Pair (p, q) is distinguishable if states $\delta(p, a)$ and $\delta(q, a)$ are distinguishable for some $a \in \Sigma$. why?

Table-filling algorithm

- Two states are distinguishable if they are not equivalent.
- Formally, two states q_1, q_2 are distinguishable, if there exists a string $w \in \Sigma^*$ such that exactly one of $\hat{\delta}(q_1, w)$ and $\delta(q_2, w)$ is an accepting state.
- Table-filling algorithm is recursive discovery of distinguishable pairs.
- Basis: Pair (p, q) is distinguishable if $p \in F$ and $q \notin F$. why?
- Induction: Pair (p, q) is distinguishable if states $\delta(p, a)$ and $\delta(q, a)$ are distinguishable for some $a \in \Sigma$. why?

TABLE-FILLING ALGORITHM:

1. DISTINGUISHABLE $= \{(p, q) \ : \ p \in F$ and $q \notin F\}$.
2. Repeat while no new pair is added
   2.1 for every $a \in \Sigma$

      add (p, q) to DISTINGUISHABLE if $(\delta(p, a), \delta(q, a)) \in$ DISTINGUISHABLE.
3. Return DISTINGUISHABLE.

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

– The proof is by contradiction.

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

- The proof is by contradiction.
- Assume that there is a pair (p, q) that is not distinguished by the algorithm, but they are not equivalent, i.e. they are indeed distinguishable (it is just that algorithm did not find them).
- Let us call such pair (p, q) a bad pair.

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

- The proof is by contradiction.
- Assume that there is a pair (p, q) that is not distinguished by the algorithm, but they are not equivalent, i.e. they are indeed distinguishable (it is just that algorithm did not find them).
- Let us call such pair (p, q) a bad pair.
- There must be a string $w \in \Sigma^*$ that distinguishes a bad pair (p, q). Let us take shortest such distinguishing string w among any bad pair, and consider corresponding bad pair (p, q).

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

- The proof is by contradiction.
- Assume that there is a pair (p, q) that is not distinguished by the algorithm, but they are not equivalent, i.e. they are indeed distinguishable (it is just that algorithm did not find them).
- Let us call such pair (p, q) a bad pair.
- There must be a string $w \in \Sigma^*$ that distinguishes a bad pair (p, q). Let us take shortest such distinguishing string w among any bad pair, and consider corresponding bad pair (p, q).
  - Notice that w can not be ϵ (Why?)

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

- The proof is by contradiction.
- Assume that there is a pair (p, q) that is not distinguished by the algorithm, but they are not equivalent, i.e. they are indeed distinguishable (it is just that algorithm did not find them).
- Let us call such pair (p, q) a bad pair.
- There must be a string $w \in \Sigma^*$ that distinguishes a bad pair (p, q). Let us take shortest such distinguishing string w among any bad pair, and consider corresponding bad pair (p, q).
  - Notice that w can not be ϵ (Why?)
  - Let w be of the form ax. Since p and q are distinguishable, we know that exactly one of $\hat{\delta}(p, ax)$ and $\hat{\delta}(q, ax)$ is accepting.
  - Then $p' = \delta(p, a)$ and $q' = \delta(q, a)$ are also distinguished by string x.

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

- The proof is by contradiction.
- Assume that there is a pair (p, q) that is not distinguished by the algorithm, but they are not equivalent, i.e. they are indeed distinguishable (it is just that algorithm did not find them).
- Let us call such pair (p, q) a bad pair.
- There must be a string $w \in \Sigma^*$ that distinguishes a bad pair (p, q). Let us take shortest such distinguishing string w among any bad pair, and consider corresponding bad pair (p, q).
  - Notice that w can not be ϵ (Why?)
  - Let w be of the form ax. Since p and q are distinguishable, we know that exactly one of $\hat{\delta}(p, ax)$ and $\hat{\delta}(q, ax)$ is accepting.
  - Then $p' = \delta(p, a)$ and $q' = \delta(q, a)$ are also distinguished by string x.
  - if (p', q') were discovered by table-filling algorithm and (p, q) must have been discovered as well.

Correctness of Table-Filling Algorithm

Theorem

If two states are not distinguished by table-filling algorithm, then they are equivalent.

- The proof is by contradiction.
- Assume that there is a pair (p, q) that is not distinguished by the algorithm, but they are not equivalent, i.e. they are indeed distinguishable (it is just that algorithm did not find them).
- Let us call such pair (p, q) a bad pair.
- There must be a string $w \in \Sigma^*$ that distinguishes a bad pair (p, q). Let us take shortest such distinguishing string w among any bad pair, and consider corresponding bad pair (p, q).
  - Notice that w can not be ϵ (Why?)
  - Let w be of the form ax. Since p and q are distinguishable, we know that exactly one of $\hat{\delta}(p, ax)$ and $\hat{\delta}(q, ax)$ is accepting.
  - Then $p' = \delta(p, a)$ and $q' = \delta(q, a)$ are also distinguished by string x.
  - if (p', q') were discovered by table-filling algorithm and (p, q) must have been discovered as well.
  - If (p', q') were not discovered by table-filling algorithm, then (p', q') is a bad pair with a shorter distinguishing string.

Minimization of DFAs

- Let A be a DFA with no unreachable state.
- Let $\equiv_A \subseteq Q \times Q$ be the state equivalence relation (computed by, say table-filling algorithm).
- Note that $\equiv_A$ is an equivalence relation.
- Let us write $[q]$ for the equivalence class of the state q.

Minimization of DFAs

- Let A be a DFA with no unreachable state.
- Let $\equiv_A \subseteq Q \times Q$ be the state equivalence relation (computed by, say table-filling algorithm).
- Note that $\equiv_A$ is an equivalence relation.
- Let us write $[q]$ for the equivalence class of the state q.
- Given a DFA A and $\equiv_A$ we can minimize A to the DFA $A_\equiv = (Q', \Sigma', \delta', q'_0, F')$, called Quotient Automata, where
  - $Q' = \{[q] \,:\, q \in Q\}$,
  - $\Sigma' = \Sigma$,
  - $\delta'([q], a) = \delta(q, a)$ for all $a \in \Sigma$,
  - $q'_0 = [q_0]$, and
  - $F' = \{[q] \,:\, q \in F\}$.

Minimization of DFAs

- Let A be a DFA with no unreachable state.
- Let $\equiv_A \subseteq Q \times Q$ be the state equivalence relation (computed by, say table-filling algorithm).
- Note that $\equiv_A$ is an equivalence relation.
- Let us write $[q]$ for the equivalence class of the state q.
- Given a DFA A and $\equiv_A$ we can minimize A to the DFA $A_\equiv = (Q', \Sigma', \delta', q_0', F')$, called Quotient Automata, where
  - $Q' = \{[q] \,:\, q \in Q\}$,
  - $\Sigma' = \Sigma$,
  - $\delta'([q], a) = \delta(q, a)$ for all $a \in \Sigma$,
  - $q_0' = [q_0]$, and
  - $F' = \{[q] \,:\, q \in F\}$.

Theorem

$A_\equiv$ is the minimum and unique (up to state renaming) DFA equivalent to A.

Proof of Minimality

Theorem

$A_\equiv$ is the minimum and unique (up to state renaming) DFA equivalent to A.

Proof.

- The proof is by contradiction.
- Assume that there is a DFA B whose size is smaller than $A_\equiv$ and accepts that same language.
- Compute equivalent states of $A_\equiv$ and B using table-filling algorithm.
- The initial states of both DFAs must be equivalent. (Why?)
- After reading any string w from their initial states, both DFAs will go to states that are equivalent. (Why?)
- For every state of $A_\equiv$ there is an equivalent state in B.
- Since the number of states of B are less than that of $A_\equiv$, there must be at least two states p, q of $A_\equiv$ that are equivalent to some state of B.
- Hence p and q must be equivalent, a contradiction.

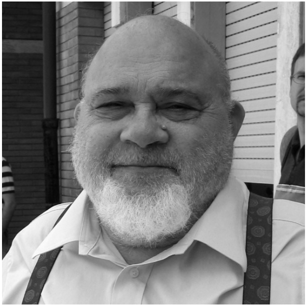
$\square$

Myhill-Nerode Theorem

- Given a languages L, two strings $u, v \in L$ are equivalent if for all strings $w \in \Sigma$ we have that $u.w \in L$ iff $v.w \in L$.
- Let $\equiv_L \subseteq \Sigma^* \times \Sigma^*$ be such string-equivalence relation.
- Note that $\equiv_L$ is an equivalence relation.
- Consider the equivalence classes of $\equiv_L$.
- When there are only finitely mane classes?

Myhill-Nerode Theorem

John Myhill



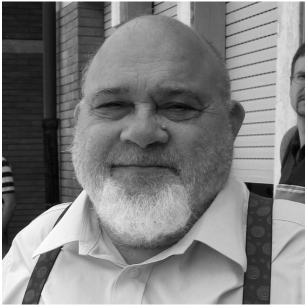
Anil Nerode

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.

Myhill-Nerode Theorem

John Myhill



Anil Nerode

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.
Moreover, the number of states in the minimum DFA accepting L is equal to the number of equivalence classes in $\equiv_L$.

Myhill-Nerode Theorem

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.

Proof.

The proof is in two parts.

- If L is regular, then a string-equivalence relation $\equiv_L$ with finitely many classes can be given by states of DFA accepting L.

Myhill-Nerode Theorem

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ *with finitely many classes.*

Proof.

The proof is in two parts.

- If L is regular, then a string-equivalence relation $\equiv_L$ with finitely many classes can be given by states of DFA accepting L. How?

Myhill-Nerode Theorem

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.

Proof.

The proof is in two parts.

- If L is regular, then a string-equivalence relation $\equiv_L$ with finitely many classes can be given by states of DFA accepting L. How?
- If there is a string-equivalence relation $\equiv_L$ with finitely many classes, one can find a DFA accepting L.

Myhill-Nerode Theorem

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.

Proof.

The proof is in two parts.

- If L is regular, then a string-equivalence relation $\equiv_L$ with finitely many classes can be given by states of DFA accepting L. How?
- If there is a string-equivalence relation $\equiv_L$ with finitely many classes, one can find a DFA accepting L. How?

□

Applying Myhill-Nerode Theorem

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.

Applying Myhill-Nerode Theorem

Theorem (Myhill-Nerode Theorem)

A language L is regular if and only if there exists a string-equivalence relation $\equiv_L$ with finitely many classes.

Equivalently,

A language L is nonregular if and only if there exists an infinite subset M of $\Sigma*$ where any two elements of M are distinguishable with respect to L.

Applying Myhill-Nerode Theorem

Theorem

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

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. The proof is by contradiction.
2. Assume that L is regular.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. The proof is by contradiction.
2. Assume that L is regular.
3. By Myhill-Nerode theorem, there is a string-equivalence relation $\equiv_L$ over L with finitely equivalence classes.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. The proof is by contradiction.
2. Assume that L is regular.
3. By Myhill-Nerode theorem, there is a string-equivalence relation $\equiv_L$ over L with finitely equivalence classes.
4. Let us consider the set of strings $\{0, 00, 000, \ldots, 0^i, \ldots\}$.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. The proof is by contradiction.
2. Assume that L is regular.
3. By Myhill-Nerode theorem, there is a string-equivalence relation $\equiv_L$ over L with finitely equivalence classes.
4. Let us consider the set of strings $\{0, 00, 000, \ldots, 0^i, \ldots\}$.
5. It must be the case that some two string 0^m and 0^n, with $m \neq n$ are mapped to same equivalence class.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. The proof is by contradiction.
2. Assume that L is regular.
3. By Myhill-Nerode theorem, there is a string-equivalence relation $\equiv_L$ over L with finitely equivalence classes.
4. Let us consider the set of strings $\{0, 00, 000, \ldots, 0^i, \ldots\}$.
5. It must be the case that some two string 0^m and 0^n, with $m \neq n$ are mapped to same equivalence class.
6. It implies that for all strings $w \in \Sigma^*$ we have that $0^m.w \in L$ iff $0^n.w \in L$.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. The proof is by contradiction.
2. Assume that L is regular.
3. By Myhill-Nerode theorem, there is a string-equivalence relation $\equiv_L$ over L with finitely equivalence classes.
4. Let us consider the set of strings $\{0, 00, 000, \ldots, 0^i, \ldots\}$.
5. It must be the case that some two string 0^m and 0^n, with $m \neq n$ are mapped to same equivalence class.
6. It implies that for all strings $w \in \Sigma^*$ we have that $0^m.w \in L$ iff $0^n.w \in L$.
7. However, $0^m 1^m \in L$ but $0^n 1^m \notin L$, a contradiction.
8. Hence L is not regular.

□

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. From Myhill-Nerode theorem, a language L is nonregular if and only if there exists an infinite subset M of $\Sigma*$ where any two elements of M are distinguishable with respect to L.

Applying Myhill-Nerode Theorem

Theorem

The language $L = \{0^n 1^n \,:\, n \geq 0\}$ is not regular.

Proof.

1. From Myhill-Nerode theorem, a language L is nonregular if and only if there exists an infinite subset M of $\Sigma*$ where any two elements of M are distinguishable with respect to L.
2. Consider the set $M = \{0^i \,:\, i \geq 0\}$.

Applying Myhill-Nerode Theorem

Theorem

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

Proof.

1. From Myhill-Nerode theorem, a language L is nonregular <u>if and only</u> if there exists an infinite subset M of $\Sigma*$ where any two elements of M are distinguishable with respect to L.
2. Consider the set $M = \{0^i \ : \ i \geq 0\}$.
3. Since any two string in M are distinguishable with respect to L (i.e. $0^m 0^n \in L$ but $0^n 1^m \notin L$ for $n \neq m$), it follows from Myhill-Nerode theorem that L is a non-regular language.

□

Some languages are not regular!

The following languages are regular or non-regular?

- The language $\{0^n 1^n \ : \ n \geq 0\}$
- The set of strings having an equal number of 0′s and 1′s
- The set of strings with an equal number of occurrences of 01 and 10.
- The language $\{ww \ : \ w \in \{0,1\}^*\}$
- The language $\{w\overline{w} \ : \ w \in \{0,1\}^*\}$
- The language $\{0^i 1^j \ : \ i > j\}$
- The language $\{0^i 1^j \ : \ i \leq j\}$
- The language of palindromes of $\{0,1\}$

Some languages are not regular!

The following languages are regular or non-regular?

- The language $\{0^n 1^n \ : \ n \geq 0\}$
- The set of strings having an equal number of $0'$s and $1'$s
- The set of strings with an equal number of occurrences of 01 and 10.
- The language $\{ww \ : \ w \in \{0,1\}^*\}$
- The language $\{w\overline{w} \ : \ w \in \{0,1\}^*\}$
- The language $\{0^i 1^j \ : \ i > j\}$
- The language $\{0^i 1^j \ : \ i \leq j\}$
- The language of palindromes of $\{0,1\}$

DFA Equivalence and Minimization

Myhill-Nerode Theorem

Pumping Lemma

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

For every regular language L there exists a constant p (that depends on L)

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

For every regular language L there exists a constant p (that depends on L) such that
for every string $w \in L$ of length greater than p,
there exists an infinite family of strings belonging to L.

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

For every regular language L there exists a constant p (that depends on L) such that for every string $w \in L$ of length greater than p, there exists an infinite family of strings belonging to L.

Why?

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

For every regular language L there exists a constant p (that depends on L) such that
for every string $w \in L$ of length greater than p,
there exists an infinite family of strings belonging to L.

Why? Think: Regular expressions, DFAs

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

For every regular language L there exists a constant p (that depends on L) such that
for every string $w \in L$ of length greater than p,
there exists an infinite family of strings belonging to L.

Why? Think: Regular expressions, DFAs Formalize our intuition!

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

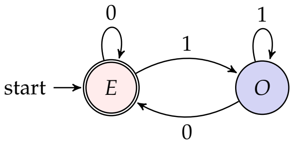
For every regular language L there exists a constant p (that depends on L) such that
for every string $w \in L$ of length greater than p,
there exists an infinite family of strings belonging to L.

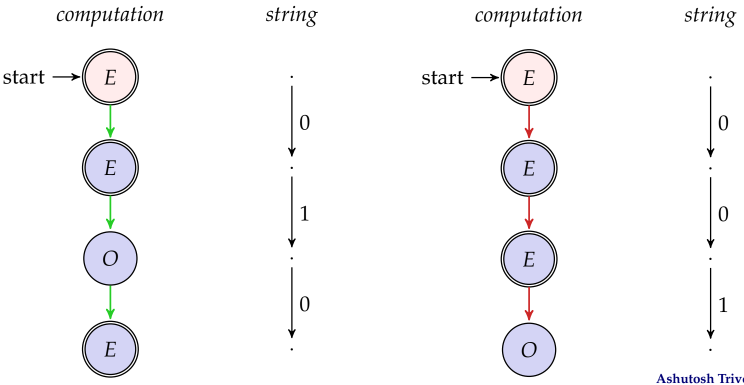
Why? Think: Regular expressions, DFAs Formalize our intuition!

If L is a regular language, then
there exists a constant (pumping length) p such that
for every string $w \in L$ s.t. $|w| \geq p$
there exists a division of w in strings x, y, and z s.t. $w = xyz$ such that

1. $|y| > 0$,
2. $|xy| \leq p$, and
3. *for all* $i \geq 0$ *we have that* $xy^iz \in L$.

A simple observation about DFA





A simple observation about DFA

Image source: Wikipedia

- Let $A = (S, \Sigma, \delta, s_0, F)$ be a DFA.
- For every string $w \in \Sigma^*$ of the length greater than or equal to the number of states of A, i.e. $|w| \geq |S|$, we have that
- the unique computation of A on w re-visits at least one state after reading first $|S|$ letters !

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

If L is a regular language, then there exists a constant (pumping length) p such that for every string $w \in L$ s.t. $|w| \geq p$ there exists a division of w in strings x, y, and z s.t. $w = xyz$ such that

1. $|y| > 0$,
2. $|xy| \leq p$, *and*
3. *for all* $i \geq 0$ *we have that* $xy^iz \in L$.

Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

If L is a regular language, then there exists a constant (pumping length) p such that for every string $w \in L$ s.t. $|w| \geq p$ there exists a division of w in strings x, y, and z s.t. $w = xyz$ such that

1. $|y| > 0$,
2. $|xy| \leq p$, *and*
3. *for all* $i \geq 0$ *we have that* $xy^iz \in L$.

- Let A be the DFA accepting L and p be the set of states in A.
- Let $w = (a_1a_2 \ldots a_k) \in L$ be any string of length $\geq p$.
- Let $s_0a_1s_1a_2s_2 \ldots a_ks_k$ be the run of w on A.
- Consider first $n + 1$ states—at least one state must occur twice.
- Let i be the index of first state that the run revisits and let j be the index of second occurrence of that state, i.e. $s_i = s_j$,
- Let $x = a_1a_2 \ldots a_{i-1}$ and $y = a_ia_{i+1} \ldots a_{j-1}$, and $z = a_ja_{j+1} \ldots a_k$.
- notice that $|y| > 0$ and $|xy| \leq n$
- Also, notice that for all $i \geq 0$ the string xy^iz is also in L.

Applying Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

$L \in \Sigma^*$ *is a regular language*
$\implies$
there exists $p \geq 1$ *such that*
for all strings $w \in L$ *with* $|w| \geq p$ *we have that*
there exists $x, y, z \in \Sigma^*$ *with* $w = xyz$, $|y| > 0$, $|xy| \leq p$ *such that*
for all $i \geq 0$ *we have that*
$xy^iz \in L$.

Applying Pumping Lemma

Theorem (Pumping Lemma for Regular Languages)

$L \in \Sigma^*$ is a regular language
$\implies$
there exists $p \geq 1$ such that
for all strings $w \in L$ with $|w| \geq p$ we have that
there exists $x, y, z \in \Sigma^*$ with $w = xyz$, $|y| > 0$, $|xy| \leq p$ such that
for all $i \geq 0$ we have that
$xy^iz \in L$.

Pumping Lemma (Contrapositive)

For all $p \geq 1$ we have that
there exists a string $w \in L$ with $|w| \geq p$ such that
for all $x, y, z \in \Sigma^*$ with $w = xyz$, $|y| > 0$, $|xy| \leq p$ we have that
there exists $i \geq 0$ such that
$xy^iz \notin L$
$\implies$
$L \in \Sigma^*$ is not a regular language.

Applying Pumping Lemma

How to show that a language L is non-regular.

1. Let p be an arbitrary number (pumping length).
2. (Cleverly) Find a representative string w of L of size $\geq p$.
3. Try out all ways to break the string into xyz triplet satisfying that $|y| > 0$ and $|xy| \leq n$. If the step 3 was clever enough, there will be finitely many cases to consider.
4. For every triplet show that for some i the string xy^iz is not in L, and hence it yields contradiction with pumping lemma.

Applying Pumping Lemma

Theorem

Prove that the language $L = \{0^n 1^n\}$ is not regular.

Applying Pumping Lemma

Theorem

Prove that the language $L = \{0^n1^n\}$ is not regular.

Proof.

1. State the contrapositive of Pumping lemma.
2. Let p be an arbitrary number.
3. Consider the string $0^p1^p \in L$. Notice that $|0^p1^p| \geq p$.

Applying Pumping Lemma

Theorem

Prove that the language $L = \{0^n 1^n\}$ is not regular.

Proof.

1. State the contrapositive of Pumping lemma.
2. Let p be an arbitrary number.
3. Consider the string $0^p 1^p \in L$. Notice that $|0^p 1^p| \geq p$.
4. Only way to break this string in xyz triplets such that $|xy| \leq p$ and $y \neq \varepsilon$ is to choose $y = 0^k$ for some $1 \leq k \leq p$.

Applying Pumping Lemma

Theorem

Prove that the language $L = \{0^n 1^n\}$ is not regular.

Proof.

1. State the contrapositive of Pumping lemma.
2. Let p be an arbitrary number.
3. Consider the string $0^p 1^p \in L$. Notice that $|0^p 1^p| \geq p$.
4. Only way to break this string in xyz triplets such that $|xy| \leq p$ and $y \neq \varepsilon$ is to choose $y = 0^k$ for some $1 \leq k \leq p$.
5. For each such triplet, there exists an i (say $i = 0$) such that $xy^i z \notin L$.
6. Hence L is non-regular.

□

Proving a language Regular

Proving Regularity

Pumping Lemma is necessary but not sufficient condition for regularity.

Proving a language Regular

Proving Regularity

Pumping Lemma is necessary but not sufficient condition for regularity.

Consider the language

$$L = \{\#a^n b^n \ : \ n \geq 1\} \cup \{\#^k w \ : \ k \neq 1, w \in \{a,b\}^*\}.$$

Verify that this language satisfies the pumping condition, but is not regular!