Nonregular languages, the Pumping Lemma, and Context-free grammars

204213 Theory of Computation

Jittat Fakcharoenphol

Kasetsart University

July 19, 2021

Outline

Short review: NFA and DFA

- For a **deterministic** finite automaton, given its current state and an input symbol from the alphabet, the next state is determined.

Short review: NFA and DFA

- For a **deterministic** finite automaton, given its current state and an input symbol from the alphabet, the next state is determined.
- For a **nondeterministic** finite automaton, given its current state and an input symbol from the alphabet, there can be many possible states (or none).

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.
- Define δ' so that M correctly simulates many copies of N.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q_0', F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.
- Define δ' so that M correctly simulates many copies of N.
- Carefully handle ε.

Proof of the NFA-DFA Equivalence

Given an NFA $N = (Q, \Sigma, \delta, q_0, F)$, we shall construct an equivalence DFA $M = (Q', \Sigma, \delta', q'_0, F')$ that recognizes the same language.

- Note that both automata take the same alphabet Σ.
- Let $Q' = \mathcal{P}(Q)$.
- Define δ' so that M correctly simulates many copies of N.
- Carefully handle ε.
- M accepts any state $R \in Q'$ such that $R \cap F \neq \emptyset$.

Definition [regular expression]

- An inductive definition of regular expressions.

Definition [regular expression]

- An inductive definition of regular expressions.
- R is a **regular expression** if R is
  1. a for some $a \in \Sigma$,
  2. ε,
  3. $\emptyset$,
  4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions,
  5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions, and
  6. (R_1^*) where R_1 is a regular expression.

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular.

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time**

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time by considering how regular expressions can be constructed.**

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time by considering how regular expressions can be constructed.**
- If a language is regular, then it can be described by a regular expression.

มี FA M ที่ recognizes L , จะมี regex ที่ describe L

Equivalence

Theorem 1

A language is regular iff some regular expression describes it.

There are two directions to prove the theorem:

- If a language is described by a regular expression, then it is regular. **Proved last time by considering how regular expressions can be constructed.**
- If a language is regular, then it can be described by a regular expression. **Quick overview last time. Recap today.**

The second part

Theorem 2

Any regular language can be described with a regular expression.

- What do we know?

The second part

Theorem 2

Any regular language can be described with a regular expression.

- What do we know?
    - A is a regular language.

The second part

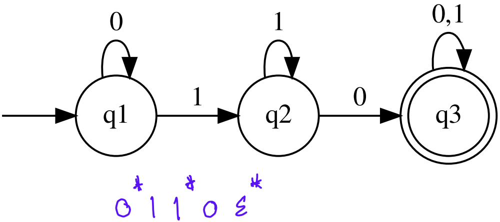
Theorem 2

Any regular language can be described with a regular expression.

- What do we know?
  - A is a regular language.
- What does that mean? → เพื่อให้ง่าย (DFA, NFA equivalent กัน)
  - There is a DFA M that recognizes A.

Practice: M_1

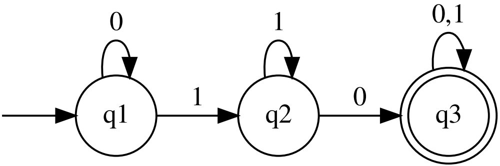
Consider the following DFA M_1.



We would like to find a regular expression describing the language reconized by M_1.

Practice: M_1

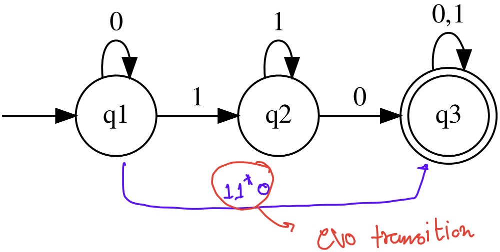
Consider the following DFA M_1.



We would like to find a regular expression describing the language reconized by M_1.
In this case, we can work out one to be $0^*11^*0\{0,1\}^*$.

Practice: M_1

Consider the following DFA M_1.



We would like to find a regular expression describing the language reconized by M_1.

In this case, we can work out one to be $0^*11^*0\{0,1\}^*$.

However, we would like to do this for any DFA. If we can show that this is possible, then we are done.

"Baby step"

- Instead of trying to convert the whole DFA to a regular expression in one step, we will try to make some progress.
- If we can always make some progress, we surely get to the finish line for sure. How? **Think about induction.**
- But what kind of progress?
    - It maybe better to start by asking what kind of finishing line that we want.

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

Goal

Simplest FA for Regular Expression construction:

- DFA (or even NFA) with two states: one start state and one accept state.

But how could we get there?
After thinking a bit it is quite straight-forward.

- Try to reduce the number of states.
- Each step decreases the number of states by one.

Note: Power-up required

- To accommodate the state reduction procedure, we have to allow transition edges with regular expressions.
- This is fine: we shall define the generalized nondeterministic finite automata.
- A **generalized nondeterministic finite automata** are nondeterministic finite automata where we allow regular expressions as labels on transition arrows.
- A GNFA can move to a new state only if it can read a block of input symbols that is described by the regular expression on the arrow.
- For everything to actually work out, we need a GNFA to be in a special form. But we leave the detail out for this course.

GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.

GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.
- If a GNFA G has more than 2 states:
  - Pick one state $q_{rip} \notin \{q_{start}, q_{accept}\}$.

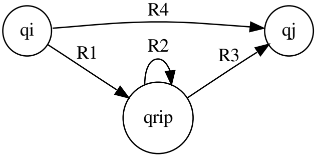
GNFA $\Rightarrow$ Regular expression

- If a GNFA G has 2 states, the conversion is straight-forward.
- If a GNFA G has more than 2 states:
  - Pick one state $q_{rip} \notin \{q_{start}, q_{accept}\}$. (There should be one, why?)

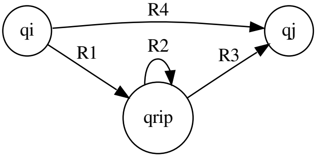
GNFA $\Rightarrow$ Regular expression

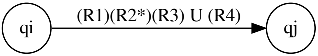
- If a GNFA G has 2 states, the conversion is straight-forward.
- If a GNFA G has more than 2 states:
  - Pick one state $q_{rip} \notin \{q_{start}, q_{accept}\}$. (There should be one, why?)
  - Build an equivalent G' by removing q_{rip}
  - Repeat.

Removing q_{rip}

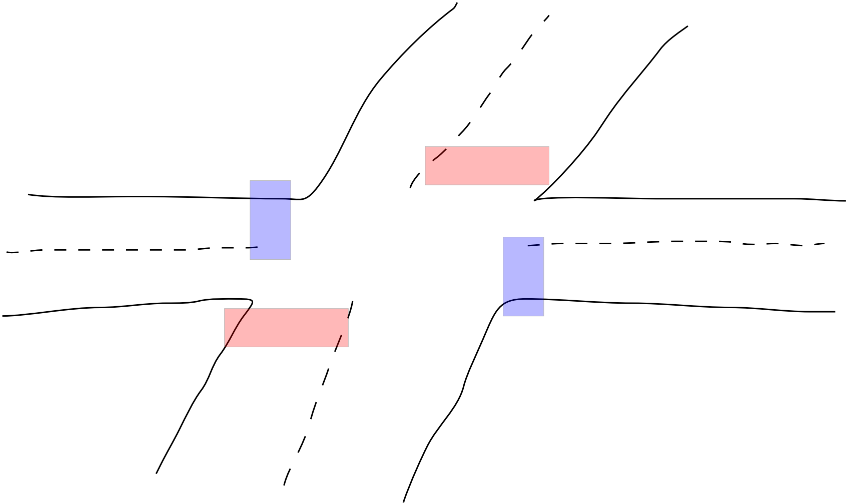


Removing q_{rip}





Traffic light control



Extracting string constants

```
#include <stdio.h>

main()
{
  int a, b;
  scanf("%d %d",&a,&b);
  printf("Hello!  \"welcome\" %d\n",a+b);
}
```

What is the limit of DFA/NFA/RegEx?

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

  - Really? I don't believe it until I (or you) have proved it.

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

  - Really? I don't believe it until I (or you) have proved it.
  - Some intuition: any DFA M recognizing B seems to have to remember the number of 0, but since M has finite state it will remember incorrectly when n is very large.

What is the limit of DFA/NFA/RegEx?

- They all have the same power. (Why?)
- Are there any languages these machines can't recognize?
  - Yes. We'll see one now:

$$B = \{0^n 1^n | n \geq 0\}.$$

  - Really? I don't believe it until I (or you) have proved it.
  - Some intuition: any DFA M recognizing B seems to have to remember the number of 0, but since M has finite state it will remember incorrectly when n is very large.
  - Again, that's **not** a proof.

Two other languages

How about these languages?

Two other languages

How about these languages?

- $C = \{w | \ w$ has an equal number of 0's and 1's $\}$.

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.

Solution:

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.

Solution: C is not regular,

Two other languages

How about these languages?

- $C = \{w|\ w$ has an equal number of 0's and 1's $\}$.
- $D = \{w|\ w$ has an equal number of occurrences of 01 and 10 as substrings $\}$

Interestingly, both seems to require remembering the lengths which can be really long.

Solution: C is not regular, but D is!

Main tool: the pumping lemma

- The pumping lemma:

  For any regular language, there is a string length, called the pumping length*, such that*

Main tool: the pumping lemma

- The pumping lemma:

  For any regular language, there is a string length, called the pumping length*, such that for any string as long as the pumping length can be "pumped".*

Main tool: the pumping lemma

- The pumping lemma:

  For any regular language, there is a string length, called the pumping length*, such that for any string as long as the pumping length can be "pumped".*

- "pumped" — the string contains a section that can be repeated any number of times while the resulting string remains in the language.

Theorem [Pumping Lemma]

Theorem 3 (Pumping lemma)

If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s maybe divided into three pieces $s = xyz$, satisfying the following conditions:

1. *for each $i \geq 0$, $x\boxed{y^i}z \in A$,* แต่ยังไม่รู้ว่าแบ่งตรงไหน
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.* → ยาวไม่เกิน p

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular.

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.

Proving that B is not regular (1)

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.
- Let $s = 0^p1^p$. We know that $s \in B$,

Proving that B is not regular (1)

- Let B be the language $\{0^n 1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.
- Let $s = 0^p 1^p$. We know that $s \in B$, and $|s| \geq p$.

Proving that B is not regular (1)

No pump |S| 0 > 1 ⇒ contradict

- Let B be the language $\{0^n1^n | n \geq 0\}$. We prove that B is not regular using the pumping lemma. We'll prove by contradiction.
- Assume that B is regular. From the pumping lemma, we know that there exists a pumping length p.
- Let $s = 0^p1^p$. We know that $s \in B$, and $|s| \geq p$.
- Now applying the pumping lemma, we have that s can be split into $s = xyz$, and for any i, xy^iz is also in B.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y?

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities. y อยู่ที่ 0
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible. → y อยู่ใน 1
- **Case 3:** $y = 0^j1^k$ for some $j > 0$ and $k > 0$. Note that in this case we'll have that $xy^2z = x0^j1^k0^j1^kz \in B$, which is, again, not possible. y อยู่ตรงกลาง

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.
- **Case 3:** $y = 0^j1^k$ for some $j > 0$ and $k > 0$. Note that in this case we'll have that $xy^2z = x0^j1^k0^j1^kz \in B$, which is, again, not possible.
- For any cases, we have reached the contradiction.

Proving that B is not regular (2)

- We know that $xyz \in B$ and $xy^iz \in B$. (That is, we can pump y)
- But what is y? Let's try all possibilities.
- **Case 1:** If $y = 0^k$ for some $k > 0$, we have that xy^2z will have more 0 than 1; thus this case is impossible.
- **Case 2:** $y = 1^k$ for some $k > 0$. Again for the same reason, this case is impossible.
- **Case 3:** $y = 0^j1^k$ for some $j > 0$ and $k > 0$. Note that in this case we'll have that $xy^2z = x0^j1^k0^j1^kz \in B$, which is, again, not possible.
- For any cases, we have reached the contradiction.
- Thus, B is not regular.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.
- Find some string s, of length at least p, such that after pumped s will not be in the language.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.
- Find some string s, of length at least p, such that after pumped s will not be in the language.
- Get the desired contradiction.

How to use the pumping lemma

The general way to proceed:

- Assume the language is regular.
- Take the pumping length p.
- Find some string s, of length at least p, such that after pumped s will not be in the language.
- Get the desired contradiction.
- **Happy!**

Practice: Language C

ให้ C เป็น language จาก pumping lemma จะได้ว่า pumping length $= p$

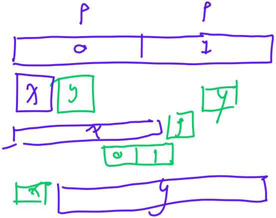
ให้ $s = 0^p 1^p$ จะได้ว่า

$s \in C$ และ $|s| \geq p$. จาก P.L. จะได้ $s = xyz$

- Let $C = \{w |\ w$ has an equal number of 0's and 1's $\}$

$s = xyz$

และ $|xy| \leq p$



Practice: Language C

non-regular

- Let $C = \{w \mid w$ has an equal number of 0's and 1's $\}$
- **Hint:** don't forget condition 3. *

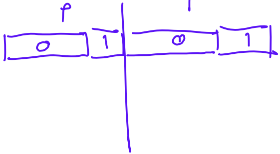
Practice: Language F

ให้ $s = 0^p 1 0^p 1$ อยู่ใน F pump lemma : p

ซึ่ง $s \in F$

string 0,1 อะไรก็ได้

- Let $F = \{ww | w \in \{0, 1\}^*\}$. พิสูจน์ว่า F ไม่เป็น regular



ปั๊ม 0 แล้วเกิน → contradict

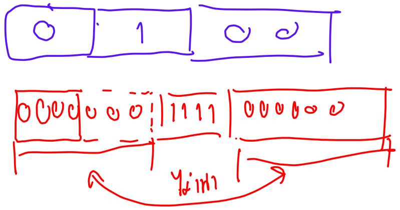
Practice: Language F

- Let $F = \{ww | w \in \{0, 1\}^*\}$.
- **Hint:** choose the right $s \in F$.

ex $G = \{0^n 1^m 0^{n+m} : n \geq 0, m \geq 0\}$

กำหนดให้ G เป็น Regular language → follow pumping lemma มี pumping length $= p$

ได้ $S = 0^p 1^p 0^{2p}$ ซึ่ง $s \in F$



$|y| \geq 1$

$|xy| \leq p$

ex $H = \{0^n 1^m ; n \geq m\}$

Proving the pumping lemma: idea (1)

- Since A is regular, we know that there exists $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A.

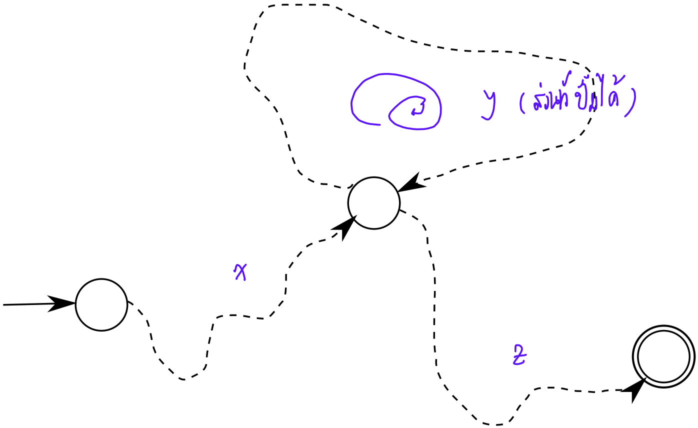
Proving the pumping lemma: idea (1)

- Since A is regular, we know that there exists $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A.
- Think about what happens when M **accepts** a really long string.

Proving the pumping lemma: idea (1)

- Since A is regular, we know that there exists $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes A.
- Think about what happens when M **accepts** a really long string.
- Since Q is finite, when taking a really long string, you'll see some state on the sequence of states from q_0 to some accept state (remember?) repeats.

Proving the pumping lemma: idea (2)



Proving the pumping lemma: steps

- Let $p = |Q|$. = จำนวน state

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state).

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p+1$ states visited by this path (including the start state). (Why do we have visited at least $p+1$ state?)

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p+1$ states visited by this path (including the start state). (Why do we have visited at least $p+1$ state?)
- Since M has p states, but we visited $p+1$ states,

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state). (Why do we have visited at least $p + 1$ state?)
- Since M has p states, but we visited $p + 1$ states, we should have visited some state twice.

Proving the pumping lemma: steps

- Let $p = |Q|$. Consider a string $s \in A$ such that $|s| \geq p$.
- Since s is accepted by M there is a path of length $|s|$ from q_0 to some state $q_f \in F$ (Why? recall the definition.)
- Consider the first $p + 1$ states visited by this path (including the start state). (Why do we have visited at least $p + 1$ state?)
- Since M has p states, but we visited $p + 1$ states, we should have visited some state twice.
- (Now you try to fill the rest.)

Quick recap: Regular languages

These sets of languages are equal:

- a set of languages recognized by deterministic finite automata,
- a set of languages recognized by nondeterministic finite automata, and
- a set of languages described by regular expressions

Quick recap: Regular languages

These sets of languages are equal:

- a set of languages recognized by deterministic finite automata,
- a set of languages recognized by nondeterministic finite automata, and
- a set of languages described by regular expressions

They are **regular languages**.

There are languages which are not regular. Today we will give you an example of languages which can be "described" by a more powerful mechanism.

An example

Grammar G_1

$$
\begin{array}{lcl}
A & \rightarrow & 0A1 \\
A & \rightarrow & B \\
B & \rightarrow & \#
\end{array}
$$

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with A

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1\\
A &\rightarrow B\\
B &\rightarrow \#
\end{aligned}
$$

Start with $A \Rightarrow 0A1$ (rule 1)

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1)

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1) $\Rightarrow 00B11$ (rule 2)

An example

Grammar G_1

$$
\begin{aligned}
A &\rightarrow 0A1 \\
A &\rightarrow B \\
B &\rightarrow \#
\end{aligned}
$$

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1) $\Rightarrow 00B11$ (rule 2) $\Rightarrow 00\#11$ (rule 3).

An example

Grammar G_1

$$
\begin{array}{lcl}
A & \rightarrow & 0A1 \\
A & \rightarrow & B \\
B & \rightarrow & \#
\end{array}
$$

ได้แบบ 0^n1^n

∴ สร้างภาษาที่ไม่ใช่ regular ได้

Start with $A \Rightarrow 0A1$ (rule 1) $\Rightarrow 00A11$ (rule 1) $\Rightarrow 00B11$ (rule 2) $\Rightarrow 00\#11$ (rule 3).

This sequence of substitution is called a **derivation**.

A grammar

From previous example, you may notice that the grammar has

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow),

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow), and
- terminals (other symbols).

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules),
- variables (symbols appearing on the left-hand side of the arrow), and
- terminals (other symbols).

To obtain a derivation, we also need a start variable.

A grammar

From previous example, you may notice that the grammar has

- a set of substitution rules (or production rules), กฎต่างๆ
- variables (symbols appearing on the left-hand side of the arrow), and ตัวแปร (ของที่จะไปถูกแทนค่าได้)
- terminals (other symbols).

To obtain a derivation, we also need a start variable. (If not specified otherwise, it is the left-hand side of the top rule.)

From the start variable

- The grammar G_1 **generates** the string 000#111.

From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:

From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:
  - Begin with start variable.

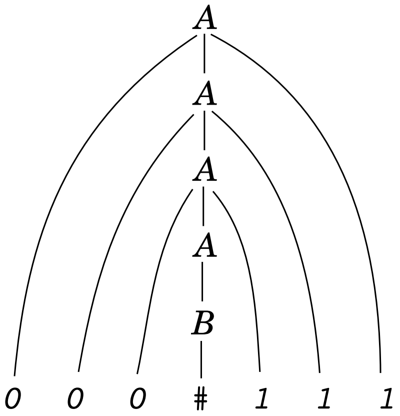
From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:
  - Begin with start variable.
  - Find a variable in the string and a rule that starts with that variable. Replace the variable with the right-hand side of the rule.

From the start variable

- The grammar G_1 **generates** the string 000#111.
- How to use the grammar to generate a string:
  - Begin with start variable.
  - Find a variable in the string and a rule that starts with that variable. Replace the variable with the right-hand side of the rule.
  - Repeat.

A parse tree



Language of the grammar

- A grammar **describes** a language by generating each string of the language.

Language of the grammar

- A grammar **describes** a language by generating each string of the language.
- For a grammar G, let $L(G)$ denote the language of G.

Language of the grammar

- A grammar **describes** a language by generating each string of the language.
- For a grammar G, let $L(G)$ denote the language of G.
- $L(G_1) =$

Language of the grammar

- A grammar **describes** a language by generating each string of the language.
- For a grammar G, let $L(G)$ denote the language of G.
- $L(G_1) = \{0^n \# 1^n | n \geq 0\}$

A context-free language

A language described by some context-free grammar is called a context-free language.

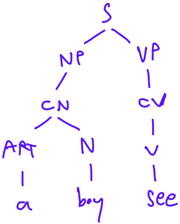
More example

Grammar G_2

$$
\begin{aligned}
S &\rightarrow NP\ VP \\
NP &\rightarrow CN | CN\ PP \\
VP &\rightarrow CV | CV\ PP \\
PP &\rightarrow PREP\ CN \\
CN &\rightarrow ART\ N \\
CV &\rightarrow V | V\ NP \\
ART &\rightarrow \texttt{a} | \texttt{the} \\
N &\rightarrow \texttt{boy} | \texttt{girl} | \texttt{flower} \\
V &\rightarrow \texttt{touches} | \texttt{likes} | \texttt{sees} \\
PREP &\rightarrow \texttt{with}
\end{aligned}
$$

Small English grammar

- Examples of strings in $L(G_2)$ are:
  - `a boy sees`



Small English grammar

- Examples of strings in $L(G_2)$ are:
  - `a boy sees`
  - `the boy sees a flower`

Small English grammar

- Examples of strings in $L(G_2)$ are:
  - `a boy sees`
  - `the boy sees a flower`
  - `a girl with a flower likes the boy`

Derivation

- Show the derivation of string "`a boy sees`".

Derivation

- Show the derivation of string "`a boy sees`".
- Try to generate more strings from G_2 and find their parse trees.

Definition [context-free grammar]

Definition

A **context-free grammar** is a 4-tuple (V, Σ, R, S), where

ไปต่อไม่ได้แล้ว

1. V is a finite set called the **variables**,
2. Σ is a finite set, disjoint from V, called the **terminals**,
3. R is a finite set of **rules**, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the **start variable**.

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv,

ให้ผลลัพธ์

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \to w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v,

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v, written as $u \stackrel{*}{\Rightarrow} v$,

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \rightarrow w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v, written as $u \overset{*}{\Rightarrow} v$,
  - if $u = v$, or

More definitions

- Let u, v, and w be strings of variables and terminals, and $A \to w$ be a rule of the grammar.
- We say that uAv **yields** uwv, denoted by $uAv \Rightarrow uwv$.
- We say that u **derives** v, written as $u \overset{*}{\Rightarrow} v$,
  - if $u = v$, or
  - if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v.$$

Example: G_3

Variable terminal start variable

$G_3 = (\{S\}, \{a, b\}, R, S)$, where R is

$$S \to aSb|SS|\varepsilon.$$

Rule

Practice

Find a CFG that describes the following language

$$\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } j = k\}$$

Example: G_4'

$G_4' = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR\}$,

Example: G_4'

$G_4' = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,

Example: G_4'

$G_4' = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,
- the rules are

$$EXPR \quad \rightarrow \quad EXPR + EXPR \mid EXPR \times EXPR \mid (EXPR) \mid a$$

Generate some string from G_4'.

Ambiguity

Find a parse tree for $a + a \times a$ in grammar G_4'.

Example: G_4

$G_4 = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR, TERM, FACTOR\}$,

Example: G_4

$G_4 = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR, TERM, FACTOR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,

Example: G_4

$G_4 = (V, \Sigma, R, EXPR)$, where

- $V = \{EXPR, TERM, FACTOR\}$,
- $\Sigma = \{a, +, \times, (,)\}$,
- the rules are

$$\begin{aligned} EXPR &\rightarrow EXPR + TERM | TERM \\ TERM &\rightarrow TERM \times FACTOR | FACTOR \\ FACTOR &\rightarrow (EXPR) | a \end{aligned}$$