Introduction to Computation Theory

Vinsong

October 19, 2025

Abstract

The lecture note of 2025 Fall Introduction to Computation Theory by professor 林智仁.

Contents

Chapter 0

Basic Knowledge

Lecture 1

2025-09-01

0.1 Mathematical Notions

0.1.1 Set & its operation

Definition 0.1.1 (Set). Omitted

Definition (Sequence & Tuple). Here are some definitions of basic containers

Definition 0.1.2 (Sequence). Sequence is the objects in order, which have two properties:

- Order:

$$(1, 2, 3) \neq (2, 1, 3)$$

- Repetition:

$$\text{Sequence} : (1, 2, 3) \neq (1, 1, 2, 3)$$

$$\text{Set} : \{1, 2, 3\} = \{1, 1, 2, 3\}$$

Definition 0.1.3 (Tuple). Finite sequence, $(1, 2, 3)$ is a 3-tuple

Definition 0.1.4 (Cartesian Product). Here is the Cartesian Product between two sets. We define

$$A = \{1, 2\},\ B = \{x, y\}$$

then,

$$A \times B = \{(1, x), (1, y), (2, x), (2, y)\}$$

0.1.2 Function & Relation

Definition 0.1.5 (Function). Function is a machine with single output.

Definition (Equivalence Relations). Here are the properties of Equivalence Relations.

Definition 0.1.6 (reflexive).

$$\forall x,\ xRx$$

Definition 0.1.7 (symmetric).

$$\forall x, y,\ xRy \iff yRx$$

Definition 0.1.8 (transitive).

$$xRy,\ yRz \implies xRz$$

Example.

$$i \equiv_7 j, \quad \text{if} \ \ 0 = i - j \mod 7$$

- Reflexive

$$i - i = 0 \mod 7$$

- Symmetric

$$i - j = 7a,\ j - i = -7a$$

- Transitive

$$i - j = 7a,\ j - k = 7b \implies i - k = 7(a + b)$$

0.1.3 String & Languages

Definition (String & Languages). Here is the definition of Language.

Example (Alphabet).

$$\{0, 1\}$$

Example (String).

$$01000$$

Definition 0.1.9 (Language). Set of Strings

$$L(A)$$

is the language of A

0.2 Definitions, Theorems, and Proofs

- **Definition**: Introduce new concept.
- **Statement**: A sentence that is either true or flase.
- **Theorem**: A statement that is true.
  - **Lemma**: A "helping" theorem.
  - **Corllary**: A theorem that follows easily from another theorem.

0.2.1 Proof by Construction

Proposition 0.2.1. Sum of degrees of every graph is even

Proof. Each edge contributes 2 nodes, so

$$\sum_{v \in V} \deg(v) = 2 \times |E|$$

Hence, the sum of degrees of every graph is even. ■

Note. The implication is the definition of graphs.

0.2.2 Proof by Contradiction

Assume the statement is false, then deduce a contradiction.

0.2.3 Proof by Induction

- Basis: Prove for $n = 0$ or $n = 1$ or some trivial case.
- Inductive Step: Assume true for $n = k$ (Induction Hypothesis), prove for $n = k + 1$.

Chapter 1

Regular Languages

1.1 Deterministic Finite Automata (DFA)

- Automaton: single
- Automata: plural

Definition 1.1.1 (Deterministic Finite Automata (DFA)). We define a DFA as a 5-tuple

$$(Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

where

- Q: Set of states (Finite)
- Σ: Alphabet (i.e. set of input characters) (Finite)
- δ: $Q \times \Sigma \to Q$: Transition Function
- $q_0 \in Q$: Start state
- $F \subset Q$: Set of accept states

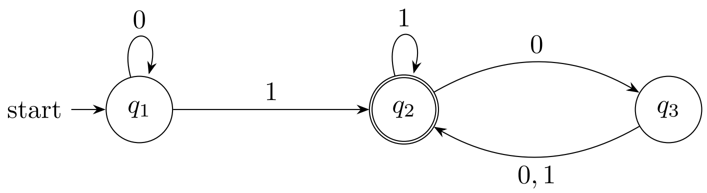


Figure 1.1: A state diagram

If we call this machine M, then we have.

$$M = (Q, \Sigma, \delta, q_0, F)$$

For the example given above,

$$Q = \{q_1, q_2, q_3\}$$
$$\Sigma = \{0, 1\}$$
$$q_0 = q_1$$
$$F = \{q_2\}$$

The δ function:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

Definition 1.1.2. The language that recognize by a Machine M is denoted as

$$L(M) = A$$

We say A is recognizeed (accepted) by M.

1.1.1 Definition of Computation

Let,

- $M = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$ be a finite automaton.
- $w = w_1, \cdots, w_n$ be a string over Σ.

Theorem 1.1.1. M accepts w if $\exists$ states $r_0 \cdots r_n$ such that

(1) $r_0 = q_0$

(2) $r_{i+1} = \delta(r_i, w_{i+1}), \quad i = [0, n-1]$

(3) $r_n \in F$

Definition 1.1.3 (Regular Language). A language is regular if recognized by some automata.

1.1.2 Regular Operations

Definition. Assume A, B are given languages,

Definition 1.1.4 (Union).
$$A \cup B = \{w \mid w \in A \vee w \in B\}$$

Definition 1.1.5 (Concatenation).

$$A \circ B = \{w_1 w_2 \mid w_1 \in A, w_2 \in B\}$$

Definition 1.1.6 (Kleene Star).

$$A^* = \{w_1 \cdots w_k \mid k \geq 0, w_i \in A\}$$

which can also be defined as

$$\bigcup_{i=1}^{\infty} A_i = \{\epsilon\} \cup A \cup A^2 \cup A^3 \cup \cdots, \quad A^0 = \{\epsilon\},\ A^n = \{wv \mid w \in A^{n-1}, v \in A\}$$

Definition 1.1.7 (closed). We say an operation R is closed if the following property holds if

$$x \in A, y \in A, \text{ then } xRy \in A$$

Theorem 1.1.2. Regular languages are closed under the union, concatenation, and Kleene star.

Proof. We define two machines as follows

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

if we union them, we can define a new machine

$$M_1 \cup M_2 = \begin{cases} M = (Q, \Sigma, \delta, q_0, F) \\ Q = \{(r_1, r_2) \mid r_1 \in Q_1, r_2 \in Q_2\} \\ \delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a)) \\ q_0 = (q_1, q_2) \\ F = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\} \end{cases}$$

Hence, regular languages are closed under union. ∎

Lecture 2

2025-09-08

1.2 Nondeterministic Finite Automata (NFA)

First, we see a NFA that accept strings with 1 in 3rd position from the end,

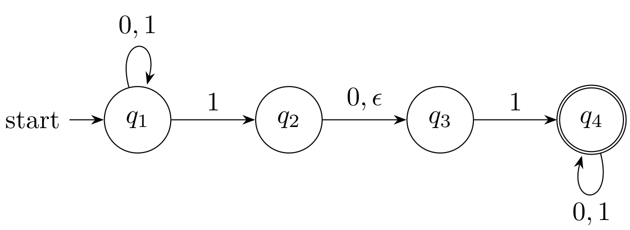


Figure 1.2: NFA machine

- δ is not a function, i.e. $\delta(q_1, 1) = q_1$ or q_2
- ϵ between q_2, q_3 means q_2 can move to q_3 without any input

We can transport NFA to DFA by some method, for example, for the above NFA we can have:

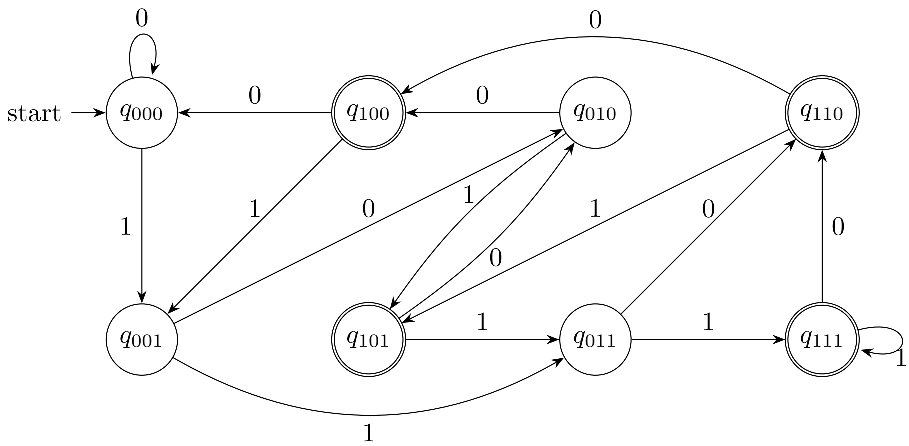


Figure 1.3: NFA machine transport to DFA

We can record it in three bits, it will be complicated.

Definition 1.2.1 (power set).

$$P(Q) = \{X | X \in Q\}$$

which contain all the $2^{|Q|}$ combinations.

Definition 1.2.2 (Nondeterministic Finite Automata (NFA)). We define a NFA as a 5-tuple

$$M = (Q, \Sigma_\epsilon, \delta, q_0, F)$$

where

- Q: Set of states (Finite)
- $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
- δ: $Q \times \Sigma_\epsilon \to P(Q)$
- $q_0 \in Q$
- $F \subset Q$

Theorem 1.2.1. We have w

$$w = y_1 \cdots y_m \quad \text{where } y_i \in \Sigma_\epsilon$$

A sequence $r_0 \cdots r_m$ such that

(1) $r_0 = q_0$

(2) $r_{i+1} = \delta(r_i, y_{i+1}), \quad i = [0, n-1]$

(3) $r_n \in F$

Note. So m may not be the original length (as y_i may be ϵ)

1.2.1 Equivalence of DFA and NFA

From DFA $\Rightarrow$ NFA. Formally DFA is not an NFA due to Σ and Σ_ϵ. but we can easily handle this by adding

$$q_i, \epsilon \to \emptyset$$

For NFA $\Rightarrow$ DFA, we have the example on the slides on a graph.

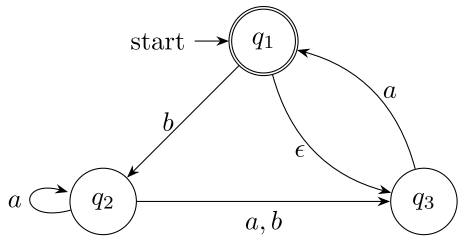


Figure 1.4: NFA example

$$\Downarrow$$

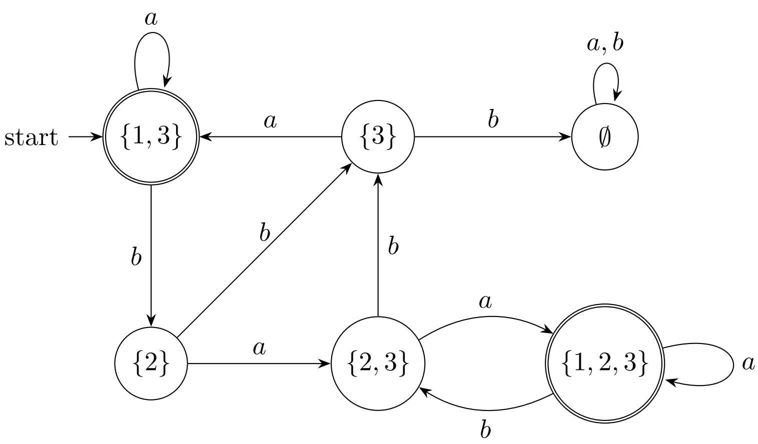


Figure 1.5: DFA convertion example

- Remove the states that are not reachable.
- Remove the states that not handle the ϵ transition. For example, the start state

$$\{q_1\}\ \text{wrong} \quad \to \quad \{q_1, q_3\}\ \text{correct}$$

Definition 1.2.3.

$$E(\{q_0\}) = \{q_0\} \cup \{\text{states reached by } \epsilon \text{ from } q_0\}$$

Then we can redefine the procedure formally.

Theorem 1.2.2. Given a NFA

$$M = (Q, \Sigma, \delta, q_0, F)$$

We can convert it to a DFA

$$M' = (Q', \Sigma, \delta', q'_0, F')$$

where

- $Q' = P(Q)$
- $q'_0 \in P(Q) = E(\{q_0\})$
- $F' = \{R \mid R \in Q', R \cap F \neq \emptyset\}$
- δ':

$$\delta'(R, a) = \bigcup_{r \in R} E(\delta(r, a))$$

1.2.2 Closure under reqular operations

We give two NFAs N_1, N_2,

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

note that $\epsilon \notin \Sigma$, and the graph of them are:

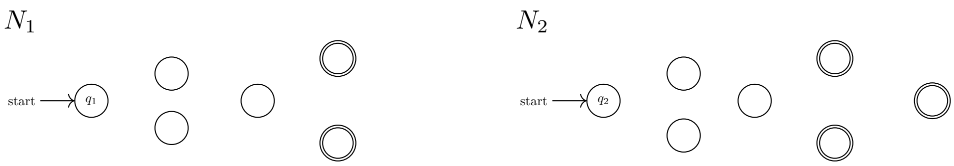


Figure 1.6: N_1, N_2

- **Union**: We can contrruct the $N_1 \cup N_2$ in

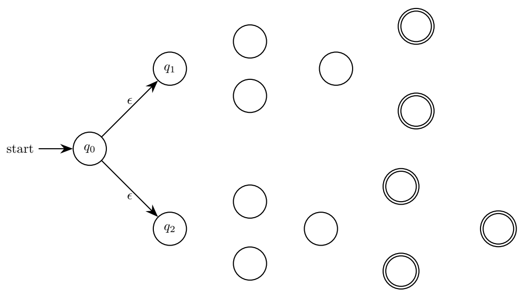


Figure 1.7: $N_1 \cup N_2$

Proposition 1.2.1 (Construction of Union). New NFA is

$$N_1 \cup N_2 = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

where

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- δ :

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0, a = \epsilon \\ \emptyset & q = q_0, a \neq \epsilon \end{cases}$$

- $F = F_1 \cup F_2$

- **Concatenation**: We can construct the $N_1 \circ N_2$ in

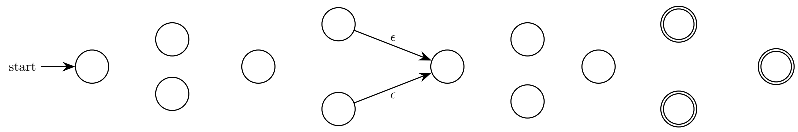


Figure 1.8: $N_1 \circ N_2$

Proposition 1.2.2 (Construction of Concatenation)**.** New NFA is

$$N_1 \circ N_2 = (Q,\ \Sigma,\ \delta,\ q_0,\ F)$$

where

- $Q = Q_1 \cup Q_2$
- δ :

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1\ F_1 \\ \delta_2(q,a) & q \in Q_2 \\ \delta_1(q,\epsilon) \cup \{q_2\} & q \in F_1, a = \epsilon \\ \delta_1(q,\epsilon) & q \in F_1, a \neq \epsilon \end{cases}$$

- $q_0 = q_1$
- $F = F_2$

- **Kleene star**: N_1^* can also accept $\{\emptyset\}$, then we can construct the N_1^* in

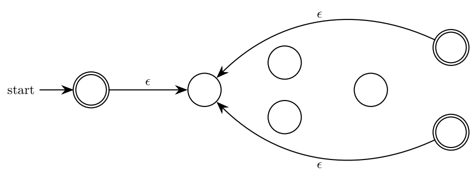


Figure 1.9: N_1^*

Proposition 1.2.3 (Construction of Kleene Star)**.** New NFA is

$$N_1^* = (Q_1,\ \Sigma,\ \delta_1,\ q_0,\ F_1)$$

where

- $Q = Q_1 \cup \{q_0\}$

- δ :

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \ F_1 \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1, a = \epsilon \\ \delta_1(q,\epsilon) & q \in F_1, a \neq \epsilon \\ \{q_1\} & q = q_0, a = \epsilon \\ \emptyset & q = q_0, a \neq \epsilon \end{cases}$$

- $F = F_1 \cup \{q_0\}$

Note. Some operations are also closed under regular languages,

- **Intersection:**

$$A_1 \cap A_2$$

Use the product automaton (the same construction as for Union). A string is accepted if and only if the state is in the accept states of both N_1 and N_2 at the same time.

- **Set Difference:**

$$A_1 - A_2$$

Use the product automaton as well. A string is accepted if the state is in the accept states of N_1 but *not* in the accept states of N_2.

- **Complement:**

$$A_1^c = \Sigma^* - A_1$$

Since Σ^* is regular and the class of regular languages is closed under set difference, A_1^c is also regular.

Lecture 3

1.3 Regular expressions

2025-09-15

A regualar expression is a tool to describe a language.

Definition 1.3.1 (Regular expressions). R is a regular expressions if it is one of the following expressions:

(1) a, where $a \in \Sigma$

(2) ϵ ($\epsilon \notin \Sigma$)

(3) $\emptyset$

(4) $R_1 \cup R_2$, where R_1, R_2 are regular expressions

(5) $R_1 \circ R_2$, where R_1, R_2 are regular expressions

(6) R_1^*, where R_1 is a regular expression

If their is no parentheses, we follow the order of:

$$\boxed{\texttt{Kleene star}} \to \boxed{\texttt{Concatenation}} \to \boxed{\texttt{Union}}$$

Remark.

$$R^+ = RR^*, \quad R^+ \cup \{\epsilon\} = R^*$$

For $\emptyset$ and ϵ, we have

- ϵ: empty string
- $\emptyset$: empty language (language without any string)

$$(0 \cup \epsilon)1^* = 01^* \cup 1^*$$
$$(0 \cup \emptyset)1^* = 01^*$$
$$\emptyset 1^* = 1^* \emptyset = \emptyset$$

Example. Here are some examples,

- Strings that start and end with the same symbol:

$$0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1$$

- $(\Sigma\Sigma)^*$: strings with even length
- $R \cup \emptyset = R$
- $R \circ \epsilon = R$
- $\emptyset^* = \{\epsilon\}$

Floating point numbers can also be represented by regular expressions. For example,

$$(+ \cup - \cup \epsilon)(DD^* \cup DD^*.D^* \cup D^*.DD^*), \text{ where } D = \{0, \ldots, 9\}$$

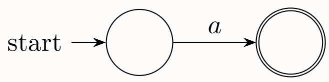
Example.

$$72 \in DD^*$$
$$2.1 \in DD^*.D^*$$
$$7. \in DD^*.D^*$$
$$.01 \in D^*.DD^*$$

Lemma 1.3.1. Language by a regular expression $\Longrightarrow$ Regular (described by an automaton)

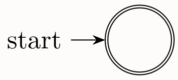
Proof. The proof is by induction,

- $R = a \in \Sigma$ can be recognize by



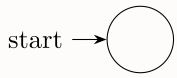
$$N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$$
$$\delta(q_1, a) = \{q_2\}$$
$$\delta(r, b) = \emptyset, r \neq q_1 \text{ or } b \neq a$$

- $R = \epsilon$



$$N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$$
$$\delta(q_1, a) = \emptyset, \forall a$$

- $R = \emptyset$



$$N = (\{q\}, \Sigma, \delta, q, \emptyset)$$
$$\delta(r, a) = \emptyset, \forall r, a$$

- $R = R_1 \cup R_2,\ R = R_1 \circ R_2,\ R = R_1^*$ have proof by NFA.

■

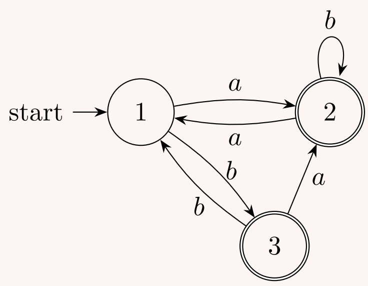
1.3.1 Convert a DFA to a regular expression

The idea is:

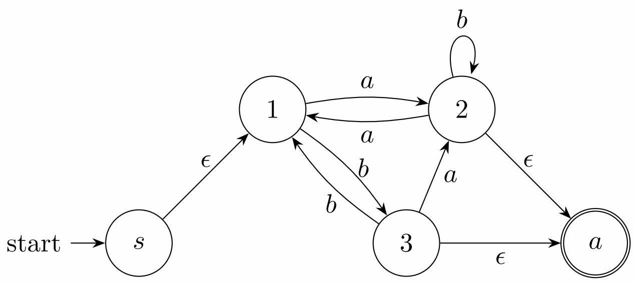
1° DFA $\longrightarrow$ GNFA

2° Remove states from GNFA until only the start and accept states.

Question. Convert the following DFA into regular expression.



Answer. First, convert to GNFA:



Next, is to remove the states one by one. We skip, so we can get the answer:

$$(a(aa \cup b)^* ab \cup b)((ba \cup a)(aa \cup b)^* ab \cup bb)^* ((ba \cup a)(aa \cup b)^* \cup \epsilon) \cup a(aa \cup b)^*$$

which is very complicated. ⊛

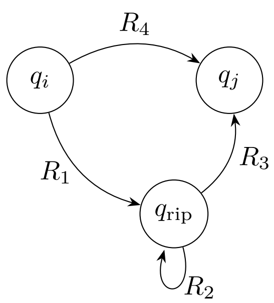
Definition 1.3.2 (Generalized NFA(GNFA)). We define a GNFA as a 5-tuple

$$G = (Q, \Sigma, \delta, q_{start}, q_{accept})$$

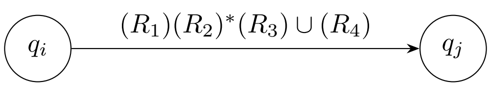
where

- F is not a se, but a single accept state q_{accept}
- δ function is:
$$(Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$$
where R is all regular expressions over Σ.
- Two new states:
$$q_{start} \to q_0 \text{ with } \epsilon$$
$$\text{any } q \in F \to q_{accept} \text{ with } \epsilon$$

Consider q_{rip} is the state being removed



The new regular expression between q_i and q_j is



We can wrote the whole process into a algorithm.

Algorithm 1.1: CONVERT(G) —State-Elimination from GNFA to RE

Input: $G = (Q, \Sigma, \delta, q_s, q_a)$ a GNFA

Output: A regular expression R for the language of G

$k \leftarrow |Q|$;

; // number of states

if $k = 2$ **then**

  return $\delta(q_s, q_a)$; // the (single) edge label from q_s to q_a

Choose any $q_{\text{rip}} \in Q \setminus \{q_s, q_a\}$;

$Q' \leftarrow Q \setminus \{q_{\text{rip}}\}$;

Initialize δ' as the restriction of δ to $Q' \times Q'$;

foreach $q_i \in Q' \setminus \{q_a\}$ **do**

  foreach $q_j \in Q' \setminus \{q_s\}$ **do**

    $R_1 \leftarrow \delta(q_i, q_{\text{rip}})$;

    $R_2 \leftarrow \delta(q_{\text{rip}}, q_{\text{rip}})$;

    $R_3 \leftarrow \delta(q_{\text{rip}}, q_j)$;

    $R_4 \leftarrow \delta(q_i, q_j)$;

    $\delta'(q_i, q_j) \leftarrow R_4 \cup \left(R_1 R_2^* R_3\right)$;

$G' \leftarrow (Q', \Sigma, \delta', q_s, q_a)$;

return CONVERT(G');

Lecture 4

2025-09-22

1.4 Pumping lemma

1.4.1 Non regular language

Some languages cannot be recognized by DFA such as,

$$\{0^n 1^n \mid n \geq 0\}$$

We might remember #0 first, but # of possible n's is ∞, so we have some method to prove that the language is non-regular.

Theorem 1.4.1 (pumping lemma)**.** If A is regular, $\exists p$ such that $\forall s \in A, |s| \geq p$,

$$\exists x, y, z, \text{ such that } s = xyz \text{ and}$$

1° $\forall i \geq 0, xy^i z \in A$

2° $|y| > 0$

3° $|xy| \leq p$

Proof. Skip, which is on the slides. ∎

1.4.2 Example for Pumping Lemma

Question. Show that the language $L = \{0^n 1^n \mid n \geq 0\}$ is not regular using the pumping lemma.

Answer. Now consider the string

$$s = 0^p 1^p$$

We know that $|s| \geq p$. By the lemma, s can be split into xyz such that

$$xy^i z \in B, \forall i \geq 0, \quad |y| > 0, \quad \text{and } |xy| \leq p$$

1° If $y = 0 \cdots 0$, then

$$xy = 0 \cdots 0 \quad \text{and} \quad z = 0 \cdots 0\, 1 \cdots 1.$$

Thus,

$$xy^2 z : \#0 > \#1.$$

Hence $xy^2 z \notin B$, a contradiction.

2° If $y = 1 \cdots 1$, then similarly

$$xy^2 z \notin B \quad \text{as} \quad \#0 < \#1.$$

3° If $y = 0 \cdots 0\, 1 \cdots 1$, then

$$xy^2 z \notin B \quad \text{since it is not of the form } 0^* 1^*.$$

> **Note.** Just pick one is sufficient to show the answer.

⊛

Question. Show that the language $C = \{w \mid \#0 = \#1\}$ is not regular using the pumping lemma.

Answer. We can use the situation in the pevious example, consider

$$s = 0^p 1^p$$

We can't proof the third condition due to $C = \{w \mid \#0 = \#1\}$ which just require the $\#0 = \#1$. Then we can use the third condition

$$|xy| \leq p$$

which means y are strict into the first 0^p we can only consider the first case.

$$|xy| \leq p \Rightarrow y = 0 \cdots 0 \text{ in } s = 0^p 1^p$$

Then,

$$xy^2 z \notin C$$

⊛

Lemma 1.4.1. When using pumping lemma, we usually use contradiction, so we use

$$\forall p\ \exists s \in A,\ |s| \geq p,\ \Big[\forall x, y, z\ \Big((s = xyz \wedge |y| > 0 \wedge |xy| \leq p)\ \rightarrow\ \exists i \geq 0,\ xy^i z \notin A\Big)\Big].$$

Use the claim and the first, second condition to get the negation of the third condition.

Question. $D = \{1^{n^2} \mid n \geq 0\}$ is not regular

Answer. We pick

$$s = 1^{p^2} \in D$$

Then, if $s = xyz, |xy| \leq p, |y| > 0$, we can get

$$p^2 < |xy^2z| \leq p^2 + p \leq (p+1)^2$$

hence, $xy^2z \notin D$. ⊛

Chapter 2

Context-Free Languages

Lecture 6

2025-10-20

2.1 Context-Free Grammars (CFG)

Which is more powerful, and can be used in compilers. A **Grammar** is a collection of substitution rules that describe the structure of a language.

> **Example.** Consider a grammar G_1:
>
> $$\begin{aligned} &A \to 0A1 \\ &A \to B \\ &B \to \# \end{aligned}$$

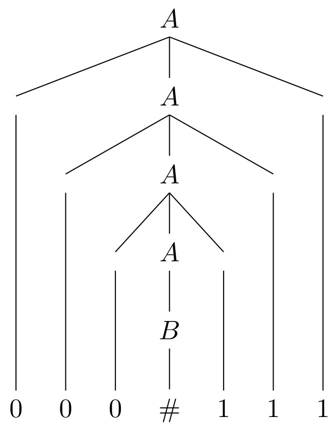
Here are the jargon terms:

- Each of one is called a **substitution rule**.
- **Variables** (non-terminals): A, B (Capital letters)
- **Terminals**: $0, 1, \#$ (Lowercase letters, numbers, symbols)
- **Start variable**: A (the variable we start with)

The process of generating strings is called **derivation**. G_1 generates $000\#111$ by

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

We can show the derivation using a **parse tree**:



2.1.1 Definition of CFG

The language of grammar G is denoted by $L(G)$, for the language we discuss here,

$$L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$$

Now we give the formal definition of CFG.

Definition 2.1.1 (Context-Free Grammar)**.** We defined a CFG as a 4-tuple

$$G = (V, \Sigma, R, S)$$

where

- V: Variables (Finite)
- Σ: Terminals (Finite)
- R: Rules:
$$\text{Variables} \rightarrow \text{Strings of Variables and Terminals (including } \epsilon)$$
- $S \in V$: Start variable

For instance, for G_1,

$$G_1 = (\{A, B\}, \{0, 1, \#\}, R, A)$$

where R is:

$$A \rightarrow 0A1 \mid B, \quad B \rightarrow \#$$

Notation. If u, v, w are strings and rule $A \rightarrow w$ is applied, then we say

$$uAv \text{ yields } uwv$$

denoted as

$$uAv \Rightarrow uwv$$

Notation. If

$$u = v \text{ or } u \Rightarrow u_1 \Rightarrow \cdots \Rightarrow u_k \Rightarrow v$$

then we write

$$v \overset{*}{\Rightarrow} u$$

Definition 2.1.2 (Language of a CFG)**.** The language generated by a CFG G with start variable S is

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

2.1.2 Examples of CFGs

Question. Consider the grammar $G_2 = (\{S\}, \{a, b\}, R, S)$:

$$S \to aSb \mid SS \mid \epsilon$$

What is $L(G_2)$?

Answer. If we let a, b be the left and right parentheses respectively, then $L(G_2)$ is the set of all balanced parentheses. ⊛

Example. Consider the grammar $G_3 = (V, \Sigma, R, S)$ where

- $V = \{\langle\text{expr}\rangle, \langle\text{term}\rangle, \langle\text{factor}\rangle\}$
- $\Sigma = \{+, \times, (,), a\}$
- R:

$$\begin{aligned}
\langle\text{expr}\rangle &\to \langle\text{term}\rangle + \langle\text{expr}\rangle \mid \langle\text{term}\rangle \\
\langle\text{term}\rangle &\to \langle\text{factor}\rangle \times \langle\text{term}\rangle \mid \langle\text{factor}\rangle \\
\langle\text{factor}\rangle &\to (\langle\text{expr}\rangle) \mid a
\end{aligned}$$

Consider the string $a + a \times a$:

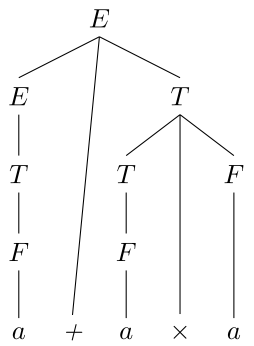


Figure 2.1: Parse tree of $a + a \times a$

Consider the string $(a + a) \times a$:

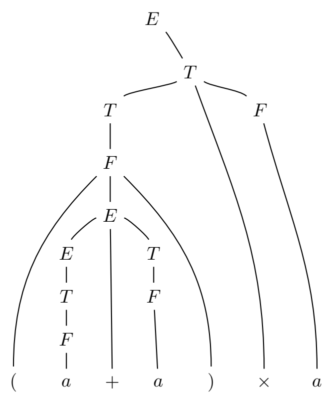


Figure 2.2: Parse tree of $(a + a) \times a$

> **Note.** The example above shows that CFGs can express operator precedence and associativity.

2.1.3 Design of CFGs

We can design CFGs in many methods. Here are some common patterns:

- Combining smaller parts:

  > **Example.** $L(G) = \{a^n b^n \mid n \geq 0\} \cup \{b^n a^n \mid n \geq 0\}$

  We can let the rule R be:

$$S_1 \to aS_1b \mid \epsilon$$
$$S_2 \to bS_2a \mid \epsilon$$
$$S \to S_1 \mid S_2$$

- From DFA:

  > **Lemma 2.1.1.** For any regular language A, there exists a CFG G such that $L(G) = A$. The rules of CFG can be
  >
  > $$R_i \to aR_j \quad \text{for each transition } \delta(q_i, a) = q_j$$
  > $$R_i \to \epsilon \quad \text{if } q_i \in F$$

  The difference is that CFG allows the format

$$R_i \to aR_jb$$

  But DFA only allows

$$R_i \to aR_j$$

  where we treat R_i as the state and let $\delta(R_i, a) = R_j$.

2.1.4 Parse Trees and Ambiguity

If we let the rules of G_3 be

$$\langle\text{expr}\rangle \to \langle\text{expr}\rangle + \langle\text{expr}\rangle \mid \langle\text{expr}\rangle \times \langle\text{expr}\rangle \mid (\langle\text{expr}\rangle) \mid a$$

We can see the following two parse trees for $a + a \times a$:

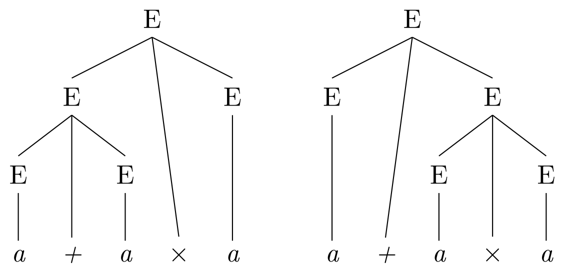


Figure 2.3: Two different parse trees for $a + a \times a$ under ambiguous grammar

This is called **ambiguity**. A CFG is **ambiguous** if there exists some string with two or more different parse trees. The above G_3 is **unambiguous**, G_3' with new rules is **ambiguous**.

However, an unambiguous grammar may also generate same parse tree but different derivations. Consider G_3:

- We can do derivation

$$\begin{aligned}\langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \\ &\Rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \times \langle \text{factor} \rangle\end{aligned}$$

- We can also do derivation

$$\begin{aligned}\langle \text{expr} \rangle &\Rightarrow \langle \text{expr} \rangle + \langle \text{term} \rangle \\ &\Rightarrow \langle \text{term} \rangle + \langle \text{term} \rangle\end{aligned}$$

which is not considered ambiguous. So we have the following definition:

Definition 2.1.3 (leftmost derivation). A **leftmost derivation** is a derivation where at each step, the leftmost variable is replaced.

Then we can have the formal definition of ambiguity:

Definition 2.1.4 (Ambiguous). w is **ambiguous** if there exists two or more different leftmost derivations.

Definition 2.1.5 (Inherent Ambiguity). A language is **inherently ambiguous** if it only has ambiguous grammars.

Example. Consider the language

$$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

We can consider the string $a^2b^2c^2$. It can be generated by two different leftmost derivations. First we consider

$$S \Rightarrow S_1 \mid S_2$$

- Using $i = j$:

$$\begin{aligned}S_1 &\to AC \\ A &\to aAb \mid \epsilon \\ C &\to cC \mid \epsilon\end{aligned}$$

  the derivation is

$$S_1 \Rightarrow AC \Rightarrow aAbC \Rightarrow aaAbbC \Rightarrow aabbC \Rightarrow aabbcC \Rightarrow aabbcc$$

- Using $j = k$:

$$\begin{aligned}S_2 &\to A'C' \\ A' &\to a\,A' \mid \epsilon \\ C' &\to b\,C'\,c \mid \epsilon\end{aligned}$$

  the derivation is

$$S_2 \Rightarrow A'C' \Rightarrow aA'C' \Rightarrow aaA'bC'c \Rightarrow aabbC'cc \Rightarrow aabbcc$$

2.2 Chomsky Normal Form