

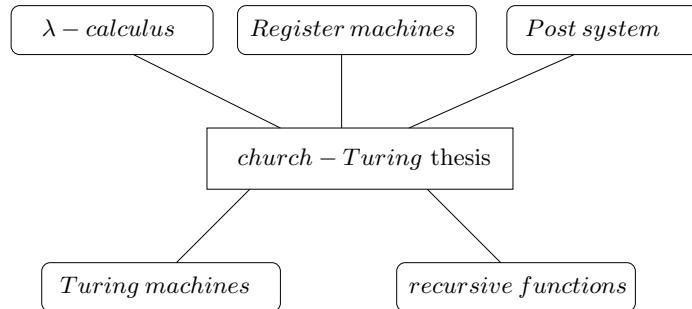
1. The study of the fundamental limits to computation

↑

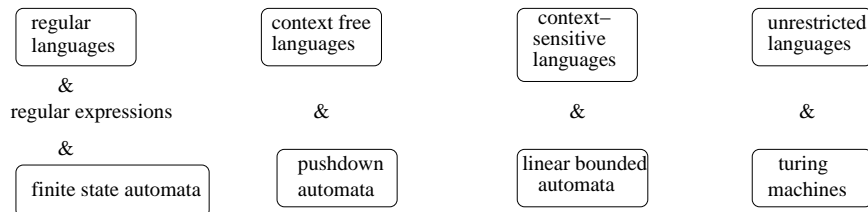
assuming unbounded but finite consumption of resources

— time , space and processing power

2. Various models of computation



3. Abstraction : Generative model *vs* machine models



STRING & SETS

Decision problems *vs* functions

A decision problem is a boolean-valued function. Set theoretically a decision problem is specified by a set A of all possible inputs and the subset $B \subseteq A$ for which the function is true.

Strings. The set of all possible inputs to a decision problem may be encoded as the set of finite-length strings over some fixed finite alphabet. All data types – numbers , graphs , trees , programs etc — can all be encoded naturally as strings. So by using the string data type we need to consider only one data type with a few basic operations.

Definition & Notations

- An alphabet is any finite set .(suitably chosen).
- The elements of an alphabet are called letters .
- A string over an alphabet Σ is any finite-length sequence of letters from Σ typically u, v, w, x, y, z are string over Σ . $|x|$ denotes the length of string x .
- There is a unique 0-length string over Σ called the *null* string or the *empty* string denoted by ε . $|\varepsilon| = 0$.

Further notations for any $a \in \Sigma$

- $a^0 \stackrel{df}{=} \varepsilon$ $a^{n+1} \stackrel{df}{=} aa^n = a^n a$
- Σ^* is the set of all (finite - length) string over Σ .
- If $\Sigma = \phi$ then $\Sigma^* = \{\varepsilon\}$ by convention. If $\Sigma \neq \phi$ then Σ^* is an infinite set of strings.

Operations on strings over Σ

- For $x, y \in \Sigma^*$, $x.y$ is the string obtained by juxtaposing y to the right of x . This is the operations of (con)catenation. Often the '.' is omitted
- (Con)catenation is
 - (i) associative : $x(yz) = (xy)z$
 - (ii) ε is the identity : $\varepsilon x = x\varepsilon = x$.
 - (iii) $|xy| = |x| + |y|$. for $x = a^m$ and $y = a^n$ we have $xy = a^m a^n = a^{m+n}$. for all $m, n \geq 0$.
- $\langle \Sigma^*, \cdot, \varepsilon \rangle$ is a monoid.
- For any $x \in \Sigma^*$,
 - $x^0 \stackrel{df}{=} \varepsilon$ $x^{n+1} \stackrel{df}{=} x^n \cdot x = x \cdot x^n$
- For $a \in \Sigma$ and $x \in \Sigma^*$,
 - $\#a(x)$ is the number of occurrences of ' a ' in x .

The prefix ordering and prefixes

Definition 1. $x \in \Sigma^*$ is a prefix of $y \in \Sigma^*$ if there exists a string $u \in \Sigma^*$ such that $x.u = y$. $x \preceq y$ denotes that x is a prefix of y .

Facts about prefixes

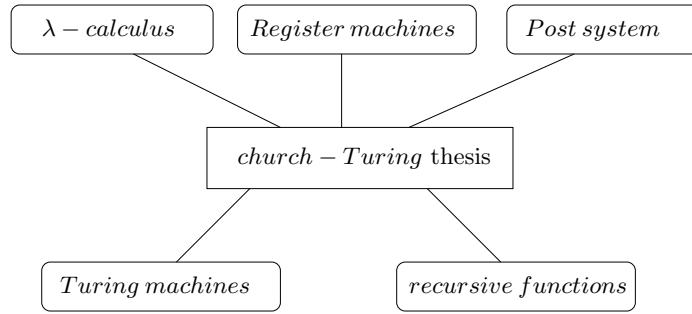
- $\varepsilon \preceq x$ for every $x \in \Sigma^*$
- $x \preceq x$ for every $x \in \Sigma^*$
- $x \preceq y$ and $y \preceq x \Rightarrow x = y$ for all $x, y \in \Sigma^*$
- $x \preceq y$ and $y \preceq z \Rightarrow x \preceq z$
- \preceq is a partial order on Σ^* .

Definition 2. x is a proper prefix of y (denoted $x \prec y$) if $x \preceq y$ and $x \neq y$.

- $x \prec y$ iff $\exists u \neq \varepsilon : x.u = y$.

Problems on strings

1. Consider the set T of trees whose nodes are labelled by letters ' a ' to ' z ' such that more than one node in a tree may have the same letter labelling it. In particular, consider the following tree. It is necessary to represent this tree as a string. Now answer the following.
 - (a) What is Σ ? Give an intuitive explanation of the letters in the alphabet Σ that you will use to represent trees such as the above.



(b) What is the representation of the above tree as a string in Σ^* ?

(c) Can you give reasons or show that

(i) Every tree in T has a unique string representation i.e no two distinct trees have the same string representation.

(ii) There are strings in Σ^* which do not represent a tree in T .

2. A string $x \in \Sigma^*$ is called a substring of another string $y \in \Sigma^*$ if for some $u, v \in \Sigma^*$, $y = u.x.v$

(a) Let $x \leq y$ if x is a substring of y . Prove that \leq is a partial order on Σ^* .

(b) Show that $x \preceq y$ implies $x \leq y$.

3. Let L be a set of labels on nodes of a graph. Give a scheme to represent both directed and undirected graphs as strings. Is it possible for the same graph to have different strings representations ?

4. If a structure $\langle A, . \rangle$ with an associative binary product operations $.$ and closed under $.$ has a left identity and a right identity then it has a unique element which is both the left identity and the left right identity.

Sets of string on Σ — The structure $\langle 2^{\Sigma^*}, \cup, \cap, \bullet, \phi, \Sigma^*, \{\varepsilon\} \rangle$

- A, B, C denote subsets of Σ^* and are called languages.
- Besides the usual set operations like \cup, \cap, \setminus and \sim (complementation) we may extend the operation of concatenation to sets of strings

- $A.B = \{x.y \mid x \in A, y \in B\}$

- $A^0 \stackrel{df}{=} \{\varepsilon\}$ $A^{n+1} \stackrel{df}{=} A^n.A = A.A^n$

↑

This ensures that $A^m.A^n = A^{m+n}$

$$A^* \stackrel{df}{=} \bigcup_{n \geq 0} A^n$$

Clearly since $A^0 \subseteq A^*$ we have $\varepsilon \in A^*$ always.

$$A^+ \stackrel{df}{=} \bigcup A^n = A.A^*$$

Note : For any $a \in \Sigma$, a^* and a^+ will be used as short hand for $\{a\}^*$ and $\{a\}^+$ respectively.
Similarly for any $x \in \Sigma^*$ x^* and x^+

Properties of set operations on languages

- \cup, \cap, \bullet are all associative operations on languages.
- \cup, \cap are also commutative (through \bullet is not)
- ϕ is the identity for \cup
 Σ^* is the identity for \cap
 $\{\varepsilon\}$ is the identity for \bullet

Note : ϕ is the empty language over Σ whereas $\{\varepsilon\}$ the (nonempty) language containing the empty string

- ϕ is the zero of \bullet
i.e $\phi.A = \phi = A.\phi$
- \cup and \cap distribute over each other
- Catenation distributes over union
 $A.(B \cup C) = (A.B) \cup (A.C)$
 $(A \cup B)C = (A.C) \cup (B.C)$
- For any indexed family (possible infinite) concatenation distributes over arbitrary unions.

$$A.\left(\bigcup_{j \in J} B_j\right) = \bigcup_{j \in J} A.B_j$$

$$\left(\bigcup_{i \in I} A_i\right).B = \bigcup_{i \in I} A_i.B$$

- The de Morgan laws hold
 $\sim (A \cup B) = \sim A \cap \sim B$
 $\sim (A \cap B) = \sim A \cup \sim B$
- $*$ satisfies the following properties

$$A^*.A^* = A^*$$

$$(A^*)^* = A^*$$

$$A^* = \{\varepsilon\} \cup A.A^* = \{\varepsilon\} \cup A^+$$

$$\phi^* = \{\varepsilon\}$$

- Why catenation does not distribute over \cap

Consider languages A, B, C and let

$$x < x' \text{ for } x, x' \in A$$

and let $y \in B - C$ and $z \in C - B$

such that $u = x.y = x'.z$

Then clearly $u \in A.B \cap A.C$

whereas $u \notin A.(B \cap C)$.

FINITE AUTOMATA & REGULAR SETS

Introduction

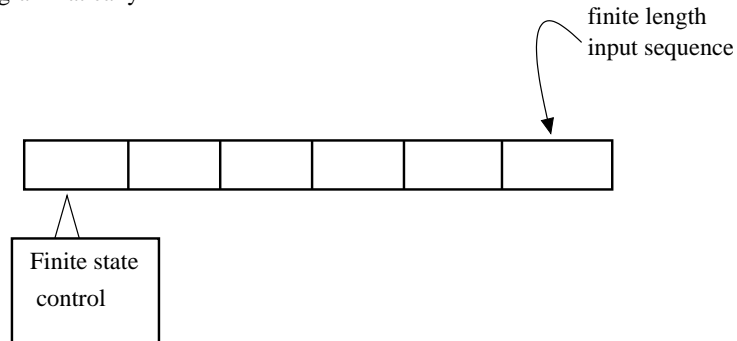
Consider a basic model of computation which we know machines could perform even the early years of the 20th century

- vending machines which “accepted ” sequence of coins of appropriate
- looms which could “read ” patterns from a finite paper tape and could replicate the patterns.
- even more recently , tape recorders and gramophone players which could “read” a finite input and convert them into sounds.
- even more recently programs that can “read” an input file containing a finite sequence of characters and take appropriate action.

All the above are concrete examples or instances of the following abstraction.

- a finite sequence of discrete inputs
- that is “read” from left to right and
- accepted by a machine with a finite number of possible states.

Diagrammatically



More abstractly we have the “finite automation” abstraction which is a structure

Definition 3. $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ a *finite-state*

automation is a structure where

- Σ is a *finite alphabet*
- Q is a *finite set of “states” of the machine*
- $q_0 \in Q$ is the *initial state*.
- $F \subseteq Q$ is the *set of final states*
- $\delta : Q \times \Sigma \longrightarrow Q$ is the *transition function which specifies for a given state and input symbol what the next state is*

Simple examples

1. Consider a vending machine which accepts only 5-rupee coins and delivers coffee for each such coin. We forget about the coffee and specify it as

δ	Σ	5
Q		
q_0		q_0

$$\Sigma = \{5\}$$

$$Q = \{q_0\}$$

$$F = \{q_0\}$$

$$\delta(q_0, 5) = q_0$$

2. Consider a machine that does not accept any other coins (say Re1 and Re2).

$$\Sigma = \{1, 2, 5\}$$

$$Q = \{q_0, q_1\}$$

$$F = \{q_0\}$$

δ	Σ	1	2	5
Q				
q_0		q_1	q_1	q_0
q_1		q_1	q_1	q_1

3. Consider a machine that accepts a nonempty sequence of Re1 coins followed by a nonempty sequence of Re2 coins but does not accept “anything else”

We abstract out the “anything else” by a new symbol \perp . So we then have

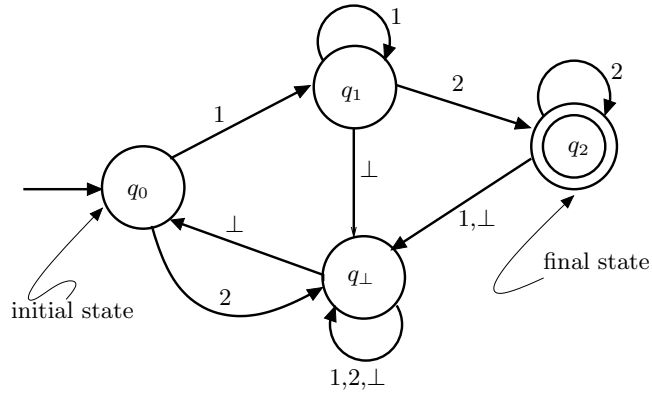
δ	Σ	1	2	\perp
Q				
q_0		q_1	q_\perp	q_\perp
q_1		q_1	q_2	q_\perp
q_2		q_\perp	q_2	q_\perp
q_\perp		q_\perp	q_\perp	q_\perp

$$\Sigma = \{1, 2, \perp\}$$

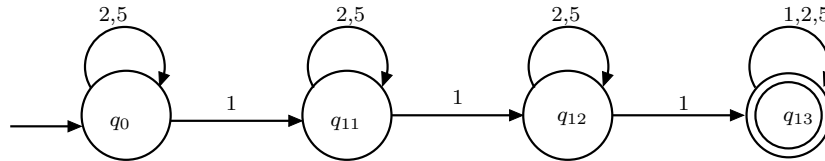
$$Q = \{q_0, q_1, q_2, q_\perp\}$$

$$F = \{q_2\}$$

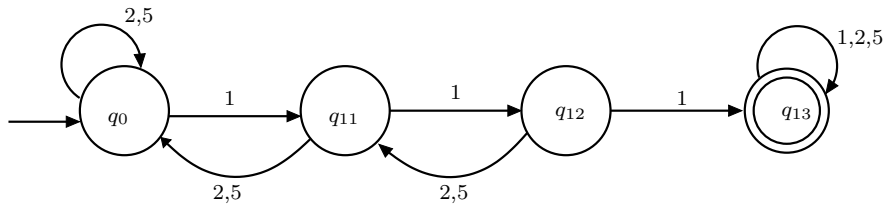
Automata may also be represented graphically



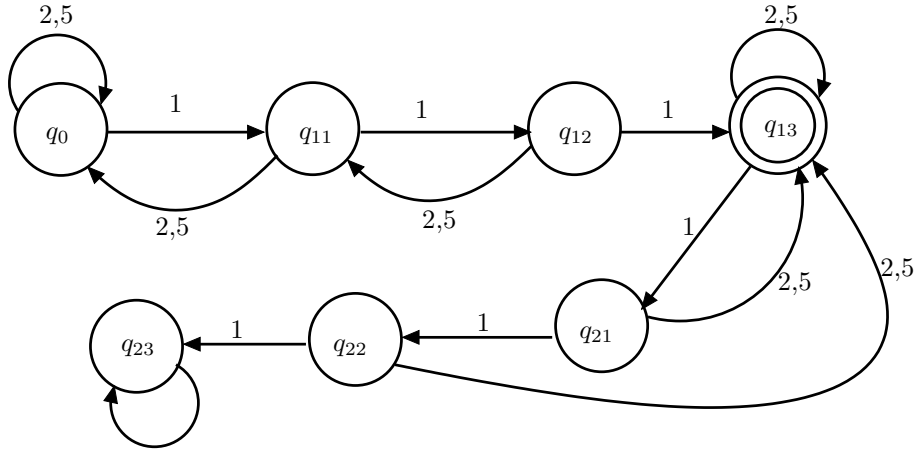
4. Consider a machine that only accepts any sequence consisting of at least 3 one-rupee coins and assume $\Sigma = \{1, 2, 5\}$



5. Consider a machine that accepts a sequence of coins in which there is at least one sub-sequence of three consecutive 1s.

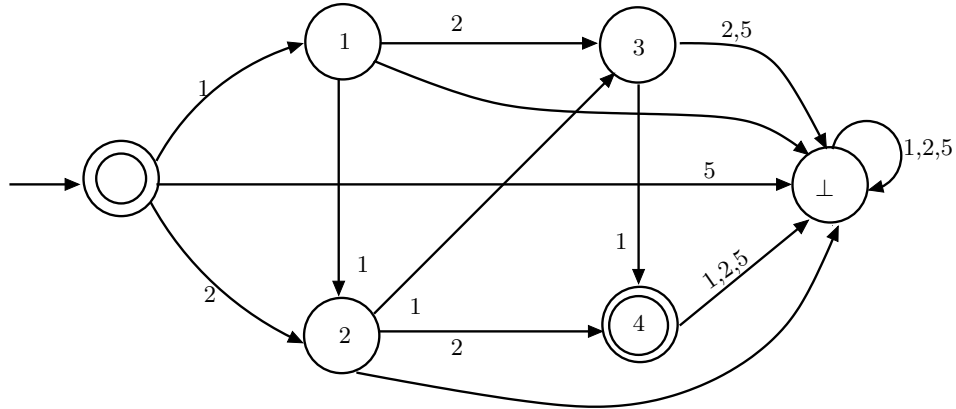


6. Consider a machine that accepts a sequence of coins in which there is exactly one sub-sequence of three consecutive 1s



7. Consider a vending machine which accepts exactly coins of denominations $\{1, 2, 3\}$ which add up to 4.

Clearly $\Sigma = \{1, 2, 5\}$ and the language accepted by the machine is $\{1^4, 121, 112, 211, 22\}$ let the states of the machine be numbered/named 0, 1, 2, 3, 4, \perp where \perp stands for any number > 4



The number on each state indicates the total value of the sequence of coins input so far.

Given a machine $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ define $\delta^* : Q \times \Sigma^* \rightarrow Q$ as the function defined by induction on the length of the input string

$$\begin{aligned} \delta^*(q, \varepsilon) &= q \\ \delta^*(q, xa) &= \delta(\delta^*(q, x), a) \end{aligned}$$

δ^* is the multi-step version of δ .

Definition 4. An automaton accepts a string $x \in \Sigma^*$ if $\delta^*(q_0, x) \in F$ and rejects it otherwise. The language accepted by it is $L(M) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\}$.

Definition 5. $A \subseteq \Sigma^*$ is regular if $A = L(M)$ for some DFA M .

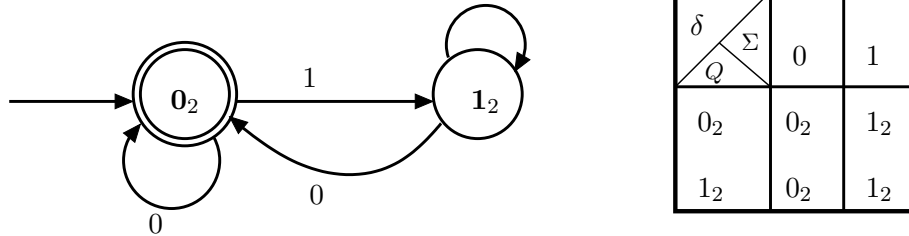
FACTS

1. ϕ and $\{\varepsilon\}$ are regular.
2. Σ^* is also regular.

3. The various languages we have defined through the various automata in the preceding examples are all regular.
4. If $L \subseteq \Sigma^*$ is regular and $\Sigma' \supseteq \Sigma$, then $L \subseteq \Sigma'^*$ is still regular.

Number-theoretic examples. Consider an automation that accepts all bit strings that are multiples of 2. Assume ε represents the number 0.

Clearly here $\Sigma = \{0, 1\}$ and the automation accepts the language $\{\varepsilon\} \cup \{x0 \mid x \in \Sigma^*\}$



Consider a machine that accepts only multiples of 3. In designing this automation we consider the following. For any $x \in \{0, 1\}^*$ let $(x)_2$ denote the integer it represents. Then we have

$$(\varepsilon)_2 = 0 \quad (0)_2 = 0 \quad (1)_2 = 1.$$

and for any $x \in \Sigma^*$ and $b \in \Sigma$

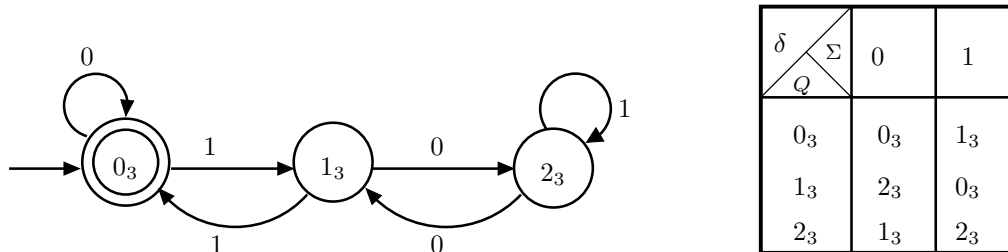
$$(xb)_2 = 2(x)_2 + b$$

The states of the automation are numbered $i \in \{0, 1, 2\}$ and satisfy the property that for any $x \in \Sigma^*$ the automation is in state $(x)_2 \bmod 3$ after consuming x . Then inductively the transition function $\delta : \{0, 1, 2\} \times \{0, 1\} \rightarrow \{0, 1, 2\}$ satisfies the property that

$$\begin{aligned} \text{if } q = (x)_2 \bmod 3 \text{ then } \delta(q, b) &= (xb)_2 \bmod 3 \\ &= (2(x)_2 + b) \bmod 3 \\ &= (2[(x)_2 \bmod 3] + b) \bmod 3 \\ &= (2q + b) \bmod 3. \end{aligned}$$

Clearly $q_0 = 0$ and $F = \{0\}$.

which gives us the following automation



Question ? How does one design a modulo 6 counter ? there are two possible answers to this question.

1. Follow the usual steps and design a 6-state automation.
2. Do a “product construction” to create the 6-state automation from the modulo 2 and modulo 3 counters.

The Product Construction

Let

$$M_1 = \langle Q_1, \Sigma, \delta_1, q_1, F_1 \rangle$$

and $M_2 = \langle Q_2, \Sigma, \delta_2, q_2, F_2 \rangle$

be automata defined on a common alphabet Σ . Their product $M_3 = M_1 \times M_2 = \langle Q_3, \Sigma, \delta_3, q_3, F_3 \rangle$ is the automaton defined by

$$Q_3 = Q_1 \times Q_2 = \{q_1', q_2' \mid q_1' \in Q_1, q_2' \in Q_2\}$$

$$F_3 = F_1 \times F_2$$

$$q_3 = (q_1, q_2)$$

$$\delta_3 : Q_3 \times \Sigma \longrightarrow Q_3 \text{ such that}$$

$$\delta((q_1', q_2'), a) = (\delta_1(q_1', a), \delta_2(q_2', a))$$

Example . The product of the automaton for the modulo 2 counter and that of the modulo 3 counter yield an automaton that accepts only bit strings that are binary representations of multiples of 6.

Lemma 1. *If $M_3 = M_1 \times M_2$ then*

$$\delta_3^*((q_1', q_2'), \varepsilon) = (q_1', q_2')$$

$$\delta_3^*((q_1', q_2'), x a) = \delta_3(\delta_3^*((q_1', q_2'), x), a)$$

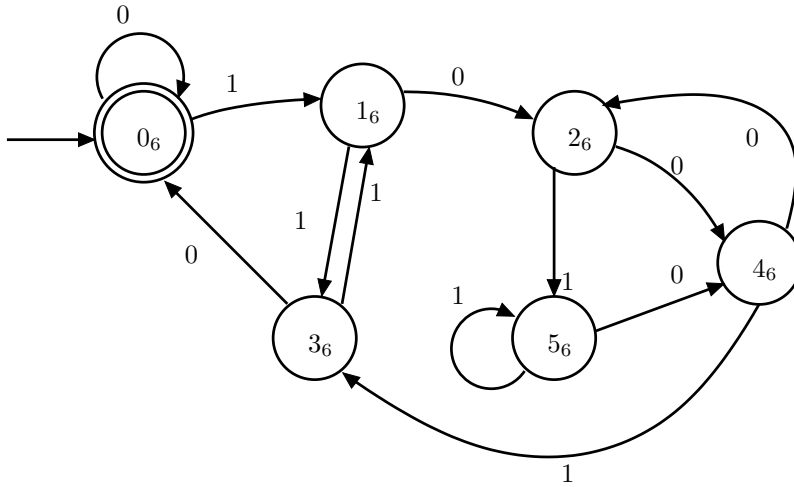
Theorem 1. $L(M_3) = L(M_1) \cap L(M_2)$

Proof. for all $x \in \Sigma^*$

$$\begin{aligned} & x \in L(M_3) \\ \Leftrightarrow & \delta_3^*((q_1, q_2), x) \in F_3 \\ \Leftrightarrow & (\delta_1^*(q_1, x), \delta_2^*(q_2, x)) \in F_1 \times F_2 \\ \Leftrightarrow & (\delta_1^*(q_1, x) \in F_1 \wedge \delta_2^*(q_2, x) \in F_2) \\ \Leftrightarrow & x \in L(M_1) \wedge x \in L(M_2) \\ \Leftrightarrow & x \in L(M_1) \cap L(M_2) \end{aligned}$$

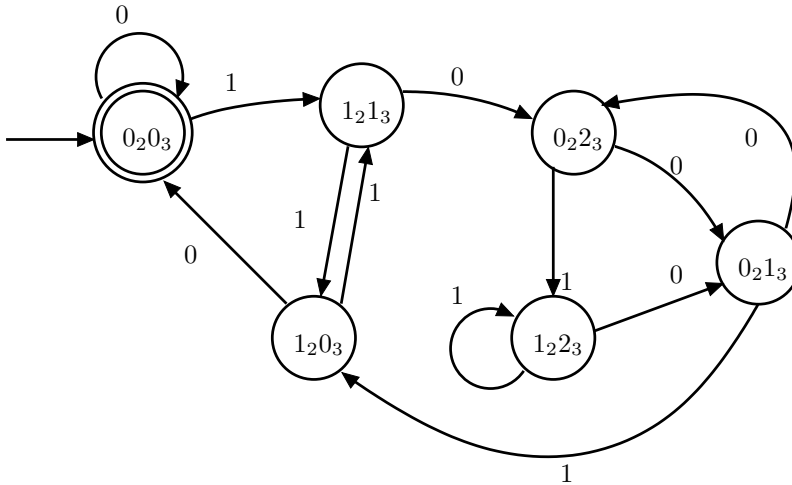
□

Example : A modulo 6 counter .
A direct construction yields.



$\delta \backslash \Sigma$	0	1
Q		
0_6	0_6	1_6
1_6	2_6	3_6
2_6	4_6	5_6
3_6	0_6	1_6
4_6	2_6	3_6
5_6	4_6	5_6

A product construction yields.



$\delta \backslash \Sigma$	0	1
Q		
0_20_3	0_20_3	1_21_3
1_21_3	0_22_3	1_20_3
0_22_3	0_21_3	1_22_3
1_20_3	0_20_3	1_21_3
0_21_3	0_22_3	1_20_3
1_22_3	0_21_3	1_22_3

The following bijection holds between the two automata states

$$i_6((i \stackrel{1 \leftrightarrow 1}{\leftarrow} \text{mod } 2)_2, (i \text{ mod } 3)_3)$$

Closure Properties of Regular languages

Let $\mathcal{R} = \{A \subseteq \Sigma^* \mid L \text{ is regular}\}$.

Theorem 2. 1. \mathcal{R} is closed under set intersection.

— follows from the previous theorem

2. \mathcal{R} is closed under complementation

i.e $A \in \mathcal{R} \Rightarrow \sim A \in \mathcal{R}$

Proof. If $A \in \mathcal{R}$, then for some DFAM $A = L(M)$. Consider the complement DFA $\tilde{M} = \langle Q, \Sigma, \delta, q_0, Q - F \rangle$ in which every non-accepting state becomes final and vice-versa. Clearly $L(\tilde{M}) = \Sigma^* - L(M) = \sim L(M) = \sim A$ □

3. \mathcal{R} is closed under union

Proof. Since \mathcal{R} is closed under both \cap and \sim it follows that for any $A, B \in \mathcal{R}$ $A \cup B = \sim(\sim A \cap \sim B)$ and hence $A \cup B$ is regular. □

4. \mathcal{R} is closed under concatenation

Proof. Let $A, B \in \mathcal{R}$ be accepted respectively by automata

$$M = \langle P, \Sigma, p_0, \delta_A, F \rangle$$

$$N = \langle Q, \Sigma, q_0, \delta_B, G \rangle$$
□

Tutorial 2 (Warm up)

0) Prove that if $\Sigma \neq \phi$ is finite Σ^* is countable.

Answer : Let $\Sigma = \{a_1, \dots, a_n\}$. Consider the mapping (1-1 and onto) of string from Σ^* to numbers in base n (without 0) such that a_i denotes the digit i in base n . The clearly with ε being mapped to 0 we have that the string in Σ^* uniquely represent numbers in base n . Hence since \mathbb{N} is countable and $\Sigma^* \xrightarrow[\text{onto}]{1-1} \mathbb{N}$ we have that Σ^* is countably infinite.

Tutorial 2

1. Given Σ is there a language that is not regular ?

Answer : Consider $\Sigma = \{a, b\}$ then the following languages are not regular

a) $\{a^n b^n \mid n > 0\}$

b) $\{x \in \Sigma^* \mid \forall u \preceq x : \#a(u) \geq \#b(u) \wedge \#a(x) = \#b(x)\}$

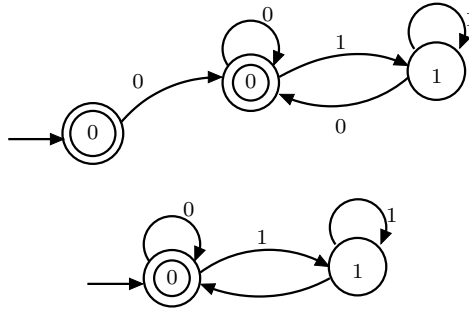
c) If $\Sigma = 0, 1$ and P is the set of strings (without leading zeroes) representing primes then P is not regular

$$P = \{x \in 1.\Sigma^* \mid (x)_2 \text{ is a prime}\}.$$

2. Given a regular language $A \subseteq \Sigma^*$ and two machines M_1 and M_2 such that $\mathcal{L}(M_1) = \mathcal{L}(M_2) = A$. Then is there an isomorphism between them ?

Answer : Not necessary as we shall in the minimization of DFAs. But a concrete example is the following. Consider a different machine to recognize multiples of 2.

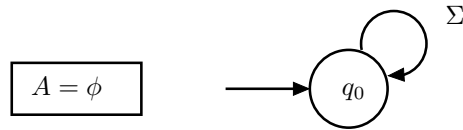
Clearly this machine has 3 states and is not isomorphic to the machine



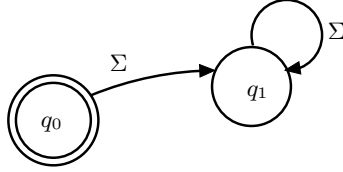
3. Prove without using the product construction or any of its consequences that every finite language on Σ is regular.

Answer : Consider a finite alphabet $\Sigma = \{a_1, \dots, a_m\}$ if $n = 0$ then clearly every language on Σ is ϕ since $\Sigma = \phi$

1) Even if $\Sigma \neq \phi$ the language A could be empty. The empty language is accepted by any automation which has no final states



If A is not empty then consider the following case $A = \{\epsilon\}$. This language is accepted by the automation.



$A \neq \{\varepsilon\}$ Initially we consider the case when $A \neq \phi$ and $\varepsilon \notin A$. If $\varepsilon \in A$ we make the initial state final. Define for each $a_i \in \Sigma$, $A_i = \{x \in \Sigma^* \mid a_i x \in A\}$. Also define $MAX(A) = \max\{|x| \mid x \in A\}$.

Basis $MAX(A) = 0$. Then $A = \{\varepsilon\}$ and we have already constructed the required automation.

IH : Assume for all B $MAX(B) < n$ for some $n > 0$ it is true that B is regular

Induction step. Let $MAX(A) = n > 0$.

Clearly $A = \bigcup_{1 \leq i \leq m} a_i.A_i$. In certain cases it is possible that $A_i = \phi$. (when a_i is not the prefix of any string in A .) Also in certain cases when $a_i \in A$, $A_i = \{\varepsilon\}$.

But since A is finite it is clear that each A_i is also finite and further $MAX(A_i) < n$ for each A_i . By the induction hypothesis there exists a machine M_i which accepts A_i .

Let $M_i = \langle Q_i, \Sigma, \delta_i, q_{i0}, F_i \rangle$ for each $1 \leq i \leq m$.

Assume for all $i \neq j$ $Q_i \cap Q_j = \phi$.

Construct the machine

$M = \langle Q, \Sigma, \delta, q_0, F \rangle$ as follows :

$$Q = \bigcup_{1 \leq i < m} Q_i \cup \{q_0\} \cup \{q_{a_i} \mid A_i = \phi\}$$

where $q_0, q_{a_1}, \dots, q_{a_m}$ are all mutually distinct and also different from each state in $\bigcup Q_i$

$F = \bigcup_{1 \leq i \leq m} F_i$ and δ is defined as follows :

for every $q_i \in Q_i, \delta(q_i, a) = \delta_i(q_i, a)$ for each $a \in \Sigma$

$$\delta(q_0, a_i) = \begin{cases} q_0 & \text{if } A_i \neq \phi \\ q_{a_i} & \text{if } A_i = \phi \end{cases}$$

and $\delta(q_{a_i}, a) = q_{a_i}$ for all $a \in \Sigma$.

NONDETERMINISTIC FINITE AUTOMATA

Introduction And Motivation

Intuitively it seems obvious that a good way to prove that the union of two regular languages is regular is to define a *DFA* that is obtained by combining the capabilities of the individual *DFAs* of the respective languages.

Example. In particular, we would like to “connect (the two *DFAs*) in parallel” to obtain an automation which can simulate either of the two behaviours.

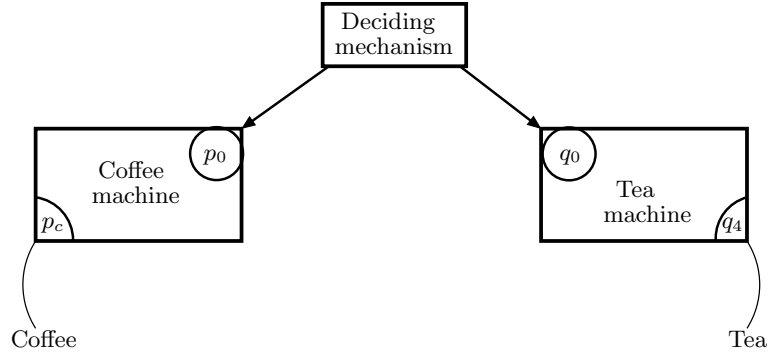
i) Consider an automation that accepts bit strings which are either multiples of 2 or multiples of 3. The resulting language is the union of the two languages.

ii) Consider a vending machine that

a) accepts a single 5-rupee coin and delivers coffee or

b) accepts any sequence of 1-rupee and 2-rupee coins that add up to 4 and delivers tea.

It should be possible to combine the two vending machine to get a single vending machine which delivers coffee or tea according to the money put in. The only thing required is a mechanism of combining the two vending machine and deciding as soon as a coin is put it whether to trigger the first machine or the second one.



A simple way to combine the two machines is as follows. Define the machine PQ with initial state pq_0

$$\left. \begin{array}{l} \delta_{PQ}(pq_0, 5) = p_c \\ \delta_{PQ}(pq_0, 1) = q_1 \\ \delta_{PQ}(pq_0, 2) = a_2 \end{array} \right| \begin{array}{l} \text{for every state in } P - \{p_0\} \\ \cup Q - \{q_0\} \text{ define} \\ \delta_{PQ}(pq, a) = \begin{cases} \delta_P(p, a) & \text{if } pq \in P \\ \delta_Q(pq, a) & \text{if } pq \in Q \end{cases} \end{array}$$

In general combining two machines M_1 and M_2 which accept languages $A, B \subseteq \Sigma^*$ bring in the union of the following disjoint sets

- 1) $A - B$
- 2) $B - A$
- 3) $A \cap B$

In the case if 1 and 2 the decision as to which machine to start is easy. However the question of which machine to run in the case of $x \in A \cap B$ requires certain external decision-making which we abstract out as non-determinism.

Having abstracted this out, it also requires that we relax the condition of acceptance or rejection as follows:

x is accepted by the combined machine if it is accepted by at least one of them

x is rejected by the combined machine only if it is rejected by both of them

Hence there should be a simple way of combining machines to produce machines that accept unions of languages such that the decisions that accept unions of languages such that the decisions mechanism is abstracted away and left to the implementor. To this end we define a “nondeterministic” automation as follows:

Definition 6. A nondeterministic finite automation (NFA) is a structure $N = \langle Q, \Sigma, \Delta, S, F \rangle$ where

$\phi \neq S \subseteq Q$ $\Delta : Q \times \Sigma \rightarrow \mathbf{2}^Q$, Δ is the transition relation. The function $\Delta^* : \mathbf{2}^Q \times \Sigma^* \rightarrow \mathbf{2}^Q$ is the natural extension of the function Δ and is defined by the rules.

$$\begin{aligned}\Delta^*(T, \varepsilon) &\stackrel{\text{df}}{=} T \\ \Delta^*(T, ax) &\stackrel{\text{df}}{=} \Delta^*(\Delta(T, a), x)\end{aligned}$$

where for any set $S \subseteq Q$

$$\Delta(T, a) = \bigcup_{q \in T} \Delta(q, a)$$

An NFA N accepts $x \in \Sigma^*$ if $\Delta^*(q_0, x) \cap F \neq \phi$ and it accepts a language $A \subseteq \Sigma^*$ if it can accept every string in A .

Intuitively a NFA accepts a string if there exists a path from one of the start states to one of the final states and it rejects a string only if there is no path from any start state to any final state. As usual $\mathcal{L}(N)$ denotes the language accepted by N .

Facts 1. Every DFA $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ is equivalent to the NFA $N = \langle Q, \Sigma, \delta, \{q_0\}, F \rangle$ with $\Delta(q, a) \stackrel{\text{df}}{=} \{\delta(q, a)\}$.

Lemma 2. For any $x, y, \in \Sigma^*$ and $T \subseteq Q$
 $\Delta^*(T, x.y) = \Delta^*(\Delta^*(T, x), y)$

Proof. By induction on $|x|$.

Basis $|x| = 0$ i.e. $x = \varepsilon$. Then

$$\Delta^*(T, \varepsilon y) = \Delta^*(T, y) = \Delta^*(\Delta^*(T, \varepsilon), y).$$

Induction step : Assume

IH. for all $|x'| \leq n$ all $T \subseteq Q$ for some $n > 0$,

$$\Delta^*(T, x'.y) = \Delta^*(\Delta^*(T, x'), y)$$

Consider $x = ax'$ with $|x| = n + 1$.

Then

$$\begin{aligned}\Delta^*(T, ax'y) &= \Delta^*(\Delta(T, a), x'y) \\ \text{let } T' = \Delta(T, a) &= \Delta^*(T', x'y) \\ \text{By IH we have} &= \Delta^*(\Delta^*(T', x'), y) \\ &= \Delta^*(\Delta^*(\Delta(T, a), x'), y) \\ \text{By def of } \Delta^* &= \Delta^*(\Delta^*(T, x), y).\end{aligned}$$

□

Lemma 3. The function Δ^* commutes with set union i.e. for any family $\{T_i \subseteq Q \mid i \in I\}$ indexed by the set I .

$$\Delta^*\left(\bigcup_{i \in I} T_i, x\right) = \bigcup_{i \in I} \Delta^*(T_i, x)$$

Proof. Again by induction on $|x|$.

Basis $x \in \varepsilon$. Then $\Delta^*\left(\bigcup_{i \in I} T_i, \varepsilon\right) = \bigcup_{i \in I} \Delta^*(T_i, \varepsilon)$

Induction step Let $x = ax'$ with $|x| > 0$

$$\begin{aligned}\Delta^*\left(\bigcup_{i \in I} T_i, ax'\right) \\ = \Delta^*\left(\Delta\left(\bigcup_{i \in I} T_i, a\right), x'\right)\end{aligned}$$

$$\begin{aligned}
\text{let } \bigcup_{i \in I} T_i = T &= \Delta^*(\Delta(T, a), x') \\
&= \Delta^*\left(\bigcup_{q \in T} \Delta(q, a), x'\right) \\
&= \Delta^*\left(\bigcup_{q \in I} \bigcup_{q_i \in T_i} \Delta(q_i, a), x'\right) \\
\text{By IH} &= \bigcup_{i \in I} \Delta^*\left(\bigcup_{q_i \in T_i} \Delta(q_i, a), x'\right) \\
&= \bigcup_{i \in I} \Delta^*(\Delta(T_i, a), x') \\
&= \bigcup_{i \in I} \Delta^*(T_i, ax') \\
&= \bigcup_{i \in I} \Delta^*(T_i, x)
\end{aligned}$$

□

The Subset Construction

Theorem 3. A NFA accepts a language $A \subseteq \Sigma^*$ iff A is regular. Equivalently a NFA accepts $A \subseteq \Sigma^*$ iff there exists a DFA that accepts A .

Proof. (\Leftarrow) Assume $A \subseteq \Sigma^*$ is regular. Then there exists a DFA $D = \langle Q_D, \Sigma, \delta_D, q_{D_0}, F_D \rangle$ that accepts A . Define $N = \langle Q_D, \Sigma, \Delta, \{q_{D_0}\}, F_D \rangle$ with $\Delta(\{q_D\}, a) = \delta(q_D, a)$.

(\Rightarrow) The proof of this part requires the construction of a DFA that accepts the same language.

Let $N = \langle Q_N, \Sigma, \Delta, S_N, F_N \rangle$ be a NFA.

Now construct a DFA

$D = \langle Q_D, \Sigma, \delta, q_{D_0}, F_D \rangle$ as follows .

$Q_D \stackrel{df}{=} 2^{Q_N}$ i.e. each state of the DFA represents a set of NFA state.

$\delta_D(q_D, a) = \Delta_N(q_D, a)$

$$= \bigcup_{q_N \in q_D} \Delta_N\{q_N\}, a)$$

$q_{D_0} = S_N$

$F_D = \{T \subseteq Q_N \mid T \cap F_N \neq \emptyset\}$

□

Lemma 4. For every $T \subseteq Q_N$ and $x \in \Sigma^*$

$$\delta_D^*(T, x) = \Delta_N^*(T, x)$$

Proof. By induction on $|x|$.

for $x = \varepsilon$ we have

$$\delta_D^*(T, \varepsilon) = T = \Delta_N^*(T, \varepsilon).$$

for $x = ax'$ we have

$$\delta_D^*(T, x) = \delta_D^*(T, ax')$$

$$= \delta_D^*(\delta_D(T, a), x')$$

$$= \delta_D^*(\Delta_N^*(T, a), x')$$

By IH $= \Delta_N^*(\Delta_N(T, a), x')$

$$= \Delta_N^*(T, ax')$$

$$= \Delta_N^*(T, x)$$

□

Claim 1. D and N accepts the same language.

Proof. For any $x \in \Sigma^*$

$$\begin{aligned} & x \in \mathcal{L}(D) \\ \Leftrightarrow & \delta_D^*(q_{D_0}, x) \in F_D \\ \Leftrightarrow & \Delta_N^*(S_N, x) \in F_D \\ \Leftrightarrow & \Delta_N^*(S_N, x) \cap F_N \neq \phi \\ \Leftrightarrow & x \in \mathcal{L}(N) \end{aligned}$$

□

End of proof of the subset construction theorem

ε -Moves : A Natural Extension of NFA

Clearly for any state $q \in Q$ of a NFA and letter $a \in \Sigma$, we have allowed $\Delta(q, a)$ to be either empty, a singleton or a set of many states. A natural extension as in the case of the vending machines is to use a non-deterministic mechanism to choose the correct vending machine which will accept the sequence of coins. This extension is called a NFA with ε -moves.

Definition 7. A NFA with ε -moves is an NFA

$$N = \langle Q_N, \Sigma, \Delta_N, S_N, F_N \rangle$$

with all components as before excepts that

$$\Delta_N : Q_N \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$$

The intention is that now it is possible for a state transition to occur without the consumption of an input. In particular now $\Delta^*(T, \varepsilon) = T$ does not necessarily hold since there may be several states in T which may allow silent moves.

Fact

Any NFA is also an NFA with ε -moves. Typically $\Delta_N(T, \varepsilon) = \phi$ in the case of a NFA without ε -moves.

Definition 8. Let $q \in Q_N$ be a state of a NFA with ε -moves. The ε -closure of q (denoted $\varepsilon^*(q)$) is the set of states reachable from q only by ε -moves and is defined inductively as follows :

$$\begin{aligned} \varepsilon^0(q) &= q \\ \varepsilon^{n+1}(q) &= \varepsilon^n(q) \cup \{q'' \in Q_N \mid \exists q' \in \varepsilon^n(q) : q'' \in \Delta_n(q', \varepsilon)\} \\ \varepsilon^*(q) &= \bigcup_{n \geq 0} \varepsilon^n(q) \end{aligned}$$

We denote the fact that $q' \in \varepsilon^*(q)$ alternately by $q \Rightarrow q'$ to mean that q' is reachable from q by only a sequence of ε -moves. Note : $q \Rightarrow q$ holds for any state

For any set $T \subseteq Q_N$ we extended this by

$$\varepsilon^*(T) = \bigcup_{q \in T} \varepsilon^*(q).$$

In other words $q' \in \varepsilon^*(T)$ iff $\exists q \in T : q \Rightarrow q'$

Definition 9. Let $\Delta_N^{\varepsilon^*} : 2^{Q_N} \times \Sigma^* \rightarrow 2^{Q_N}$ be defined as

$$\begin{aligned} \Delta_N^{\varepsilon^*}(T, \varepsilon) &= \varepsilon^*(T) = \{q' \mid \exists q \in T : q \Rightarrow q'\}. \\ \Delta_N^{\varepsilon^*}(T, ax) &= \{q \in Q_N \mid \exists q \in T : \exists q_1, q_2 \in Q_N : q_0 \Rightarrow q_1 \xrightarrow{a} q_2 \Rightarrow q_2 \wedge q \in \Delta_N^{\varepsilon^*}(q_2, x)\}. \end{aligned}$$

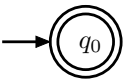
Note. $\Delta_N^*(q, \varepsilon) = q$ whereas $\Delta_N^{\varepsilon^*}(q, \varepsilon) = \varepsilon^*(q)$

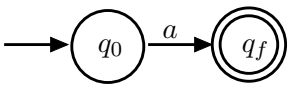
- Regular Expressions : Representation of regular languages.
- Building NFA_ε from regular expression.

Theorem 4. Every regular expression represents a regular language.

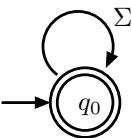
Proof. By induction on the structure of regular expressions.

Basis. ϕ denotes the empty language and is accepted by a *DFA/NFA* with no final states.

ε is accepted by  which is a *NFA*.

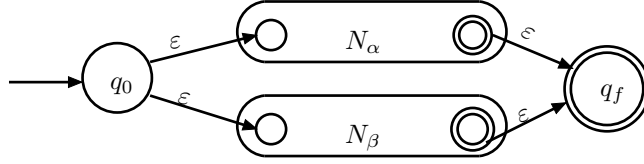
$a \in \Sigma$ is accepted by 

? is matched by any symbol of Σ obtained by a finite union of the *NFA*'s for each $a \in \Sigma$

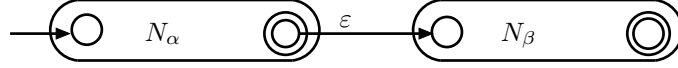
@ is any string in Σ^* is accepted by 

Induction Step

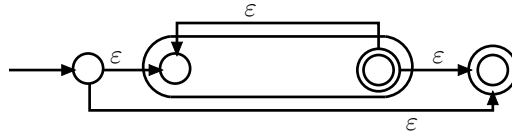
$\alpha \cup \beta$



$\alpha\beta$



α^*



□

Theorem 5. For every NFA_ϵ there exists a NFA without ϵ -moves which accepts the same language.

Proof. Let $N_\epsilon = \langle Q_\epsilon, \Sigma, \delta_\epsilon, S_\epsilon, F_\epsilon \rangle$ be a NFA with ϵ -moves. Construct the NFA $N = \langle Q, \Sigma, \delta, S, F \rangle$ without ϵ -moves as follows.

For every $q \in Q_\epsilon$ and $a \in \Sigma$

$$\boxed{q \xrightarrow{a} q' \delta \text{ iff } \exists q_1, q_2 : q \xrightarrow{\epsilon} q_1 \xrightarrow{a} q_2 \xrightarrow{\epsilon} q' \text{ in } N_\epsilon.}$$

Claim 2. N and N_ϵ accept the same language.

Proof. $x \in \mathcal{L}(N_\epsilon)$ with $x = a_1 \dots a_n$.

$$\Leftrightarrow \exists q_0 \in S \exists q_f \in F : q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_f \text{ in } N_\epsilon$$

$$\Leftrightarrow \exists q_0 \in S \exists q_f \in F : q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} q_f \text{ in } N$$

□

□

Theorem 6. Every regular language has a regular expression representation. We have already proved the following viz :

1. Every regular expression represents a regular language where the regular expressions were defined by the following

0) For every $a \in \Sigma$, \mathbf{a} is a RE and $\mathcal{L}(\mathbf{a}) = \{a\}$.

1) ϵ and ϕ are regular expression and $\mathcal{L}(\epsilon) = \{\epsilon\}$

$$\mathcal{L}(\phi) = \phi$$

2) For regular expressions α, β .

$\alpha.\beta$ is a regular expression representing $\mathcal{L}(\alpha).\mathcal{L}\beta$

$\alpha \cup \beta$ is a regular expression representing $\mathcal{L}(\alpha) \cup \mathcal{L}\beta$

- 3) For any regular expression α ,
- (α) is a regular expression representing $\mathcal{L}(\alpha)$
 - α^* is a regular expression representing $(\mathcal{L}(\alpha))^*$
 - $\sim \alpha$ is a regular expression representing $\sim \mathcal{L}(\alpha)$

Precedence of Operators

- ★ is the highest \cup is the lowest
- precedes \cup but is lower than any unary operator

Theorem 7. For every regular language $A \subseteq \Sigma^*$, there exists a regular expression α with $\mathcal{L}(\alpha) = A$.

Proof. Consider a regular language A and some NFA $N = \langle Q, \Sigma, \Delta, S, F \rangle$ which accepts A .

Construction of a regular expression $\alpha_{p,q}^X$ for any $X \subseteq Q$ and states $p, q \in Q$ such that

$$\mathcal{L}(\alpha_{pq}^X) = \{x \in \Sigma^* \mid p \xrightarrow{x} q \text{ with all intermediate states in } X\}$$

↑

If $x = a_1 a_2 \dots a_m$ then $\exists r_1, r_2, \dots, r_{m-1} \in X$:

$$p \xrightarrow{a_1} r_1 \xrightarrow{a_2} r_2 \longrightarrow \dots \xrightarrow{a_{m-1}} r_m \xrightarrow{a_m} q.$$

¹ p, q may or may not be in X

By induction on the size of X

Basis $X = \phi$. Let $a_1, a_2, \dots, a_k \in \Sigma$ be all the letters such that $p \xrightarrow{a_i} q$ for $1 \leq i \leq k$. Then

$$\text{if } p \neq q \quad \alpha_{pq}^\phi \stackrel{df}{=} \begin{cases} a_1 \cup a_2 \cup \dots \cup a_k. & \text{if } k > 0 \\ \phi & \text{if } k = 0. \end{cases}$$

$$\text{else if } p = q, \quad \alpha_{pq}^\phi \stackrel{df}{=} \begin{cases} a_1 \cup a_2 \cup \dots \cup a_k \cup \varepsilon & \text{if } k > 0 \\ \varepsilon & \text{if } k = 0. \end{cases}$$

Induction step $X \neq \phi$

Choose any $r \in X$. Let $Y = X - \{r\}$

Then $\alpha_{pq}^X \stackrel{df}{=} \alpha_{pq}^Y \cup \alpha_{pr}^Y (\alpha_{rr}^Y)^* \alpha_{rq}^Y$

Hence by this induction process for each start state $s \in S$ and final state $f \in F$ we have an expression α_{sf}^Q which represents the set of all paths from s to f in N . Hence $\bigcup_{s \in S} \alpha_{sf}^Q$ is the regular expression denoting the language accepted by N . □

Note. This theorem essentially tells us that the language RE_Σ of regular expressions over Σ is complete and that it is always possible to convert a *DFA* or a *NFA* into a regular expression.

Kleene Algebra of Regular Expressions

<ul style="list-style-type: none"> + is associative & commutative ϕ is the identity element for + + is idempotent
<ul style="list-style-type: none"> • is associative ε is the identity for • ϕ is the zero for • • distributes over + (both left & right)

$$\alpha^* = \varepsilon + \alpha\alpha^* = \varepsilon + \alpha^*\alpha$$

The equation $x = \alpha x + \beta$ has the least solution $\alpha^*\beta$

Better still define \leq on regular expression such that

$$\alpha \leq \beta \text{ iff } \alpha + \beta = \beta$$

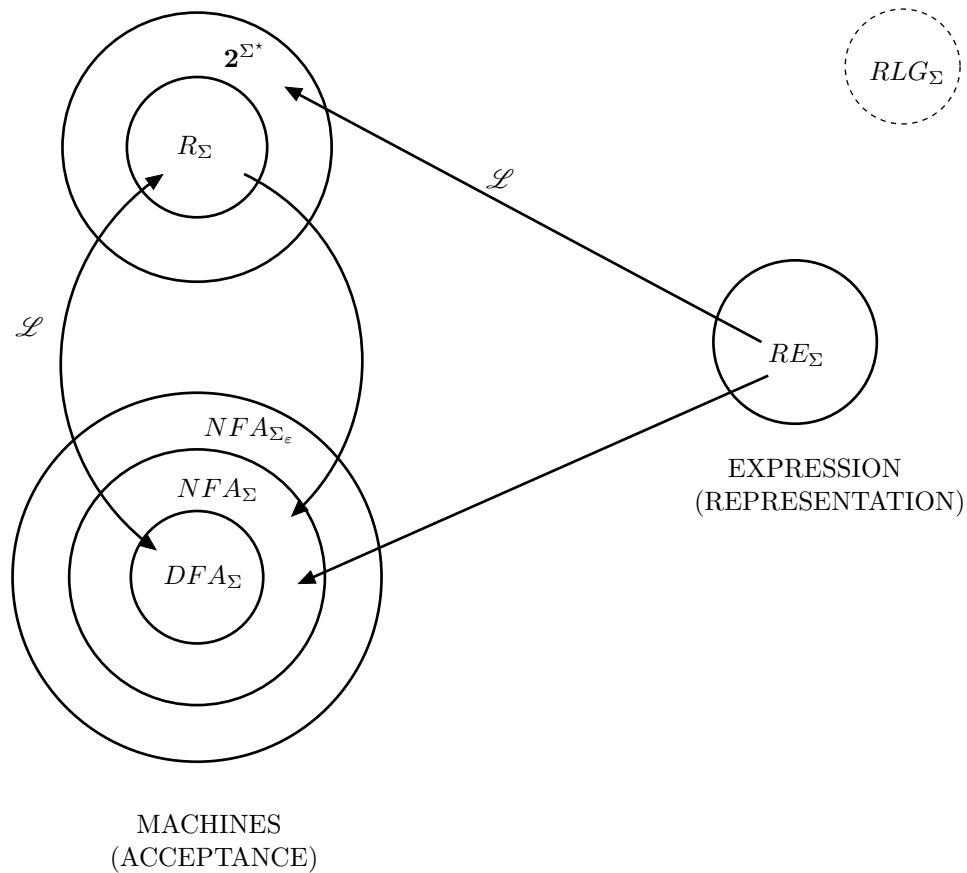
and have the two influence rules

$$\alpha\gamma + \beta \leq \gamma \Rightarrow \alpha^*\beta \leq \gamma$$

$$\text{and } \gamma\alpha + \beta \leq \gamma \Rightarrow \alpha^\alpha\beta \leq \gamma$$

The story so far

From regular expressions to regular grammars



Consider the regular expression equation

$$X = \mathbf{a}X + \mathbf{b}$$

Whose solution is the regular expression

$$\boxed{X = \mathbf{a}^*\mathbf{b}}$$

which may be obtained as the maximal fixpoint under \subseteq of the composition of monotonic functions \bullet and $+$ on the lattice 2^{Σ^*}

String Generation through Grammars

The generative process may intuitively be looked upon as follows. Consider a typical solution of the equation

$$X = A.X \cup B$$

on language over Σ^* i.e. $A, B \subseteq \Sigma^*$ and we need to find the unknown X . We may construct an increasing sequence of sets X_i , $i \geq 0$

$$\phi = X_0 \subseteq X_1 \subseteq \dots X_i \subseteq X_{i+q} \subseteq \dots$$

such that

$$X_{i+1} = A.X_i \cup B$$

The maximum solution is the set $\bigcup_{i \geq 0} X_i$

Essentially we are unfolding the recursion in the equation inductively to obtain.

$$\begin{aligned} X_1 &= A.\phi \cup B = B \\ X_2 &= A.X_1 \cup B = B \cup A.B \\ X_3 &= A.X_2 \cup B = B \cup A.B \cup A.A.B \\ &\vdots \end{aligned}$$

This allows us to “generate” a larger and larger collection of strings from the empty sets. We use this essentially to define the notion of generation by “orienting” the ‘=’ in a left-to-right fashion.

Grammers & Generation

Definition 10. A right-linear grammar over an alphabet Σ is a structure $G = \langle V, \Sigma, S, P \rangle$ where

- $V \neq \phi$ is a finite set of variables or nondeterminals
- Σ is finite alphabet with $V \cap \Sigma = \phi$.
- $S \in V$ is the start symbol of the grammar or rules
- $P \subseteq_f (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ is a finite collection of productions or rules such that each production is of only one of the following forms.

$$\left. \begin{array}{l} (i) A \longrightarrow xB \\ (ii) A \longrightarrow x \end{array} \right\} \text{ where } x \in \Sigma^* \text{ and } A, B \in V$$

i.e $p \subseteq_f V \times \Sigma^* \bullet (V \cup \{\varepsilon\})$

A derivation of a (right linear) grammar is a sequence of strings in $(\Sigma \cup V)^*$ starting from S . such that

Example. $G = \langle \{S\}, \{a, b\}, S, P \rangle$ where P consists of the productions $S \longrightarrow abS$ and $S \longrightarrow \varepsilon$.
Its derivations are of the form $S \Rightarrow^* (ab)^n S \Rightarrow (ab)^n$
 $S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow ababab$

for any string $xA \Rightarrow yB$ iff for some $z \in \Sigma^*$, $y = xz$ and $A \longrightarrow zB \in P$.

The reflexive - transitive closure of \Rightarrow is denoted \Rightarrow^* and the language generated by a grammar G with start symbol S is the set

$$\mathcal{L}(G) = \{ x \in \Sigma^* \mid S \Rightarrow^* x \}.$$

i.e $x \in \mathcal{L}(G)$ iff there exists a derivation of x from S .

In the example above

$$\varepsilon, ab, abab, \dots, (ab)^n, \dots \in \mathcal{L}(G)$$

In fact $\mathcal{L}(G) = \{ (ab)^n \mid n \geq 0 \}$

There also exists infinite derivations which do not contribute to the language generated. An infinite derivation is obtained by never applying the production $S \longrightarrow \varepsilon$.

Lemma 5. Any derivation of a right linear grammar is either of the form

$$S_0 \Rightarrow x_1 S_1 \Rightarrow \dots \Rightarrow x_1 x_2 \dots x_i S_i \Rightarrow \dots$$

or $S_0 \Rightarrow x_1 S_1 \Rightarrow \dots \Rightarrow x_1 x_2 \dots x_i S_i \Rightarrow \dots \Rightarrow x_1 x_2 \dots x_{n-1} S_{n-1} \Rightarrow x_1 x_2 \dots x_{n-1} x_n$

where $S_0, S_1, \dots, \in V$ and $x_1, x_2, \dots \in \Sigma^*$.

Theorem 8. *If G is a right linear grammar then $\mathcal{L}(G)$ is a regular language.*

Proof. Let $G = \langle V, \Sigma, S_0, P \rangle$ be a right linear grammar with $V = \{S_0, S_1, \dots, S_m\}$ with a finite collection of productions

$$P = \{ S_i \longrightarrow x_k S_j \mid S_j \in V \cup \{\varepsilon\}, S_i \in V, x_k \in \Sigma^*, 1 \leq k \leq n \}$$

If $y \in \mathcal{L}(G)$ then there exists a derivation $S_0 \Rightarrow^* y$ of the form

$$\boxed{S_0 \Rightarrow x_{i_1} S_{i_1} \Rightarrow x_{i_1} x_{i_2} S_{i_2} \Rightarrow \dots \Rightarrow x_{i_1} x_{i_2} \dots x_{i_p} = y}$$

with $\{S_{i_1}, S_{i_2}, \dots, S_{i_p}\} \subseteq V$.

We construct an automation (*NFA*) with a start state labelled S_0 . For each production

$$S_i \longrightarrow a_1 \dots a_l S_j \text{ where } S_j \in V \cup \{\varepsilon\}$$

Construct states $S_{i_j}, S_{i_2}, \dots, S_{i_{l-1}}, S_{i_l}$ such that

$$S_{i_j} \xrightarrow{a_j} S_{i_{j+1}} \text{ for } 1 \leq j \leq l$$

is a transition in Δ . If $S_j \neq \varepsilon$ then $S_{i_j} = S_j$ otherwise S_{i_l} is a new final state. Clearly for each such production.

We have the following claim

$$\boxed{S_i \longrightarrow a_1 \dots a_l S_{i_l} \text{ for } S_{i_l} \in V \cup \{\varepsilon\} \text{ iff } \Delta^*(S_i, a_1 \dots a_l) = S_{i_l}}$$

The following claim may then be proven.

$$\boxed{S_0 \Rightarrow_0 y \text{ iff } \Delta^*(S_0, y) \in F}$$

where F is the collection of final states of the automation.

It follows therefore that the language generated by a right linear grammar is also the language accepted by some *NFA* and hence must be regular. \square

Theorem 9. *Every regular language over Σ may be generated by a right linear grammar.*

Proof. Let $A \in R_\Sigma$. Then there exists a *DFA*

$$D = \langle Q, \Sigma, \delta, q_0, F \rangle$$

which accepts A . Construct a grammar

$$G = \langle V, \Sigma, q_0, P \rangle \text{ where } V = Q$$

with

$$\boxed{q_i \longrightarrow a q_j \text{ iff } \delta(q_i, a) = q_j}$$

and

$$\boxed{q_i \longrightarrow \varepsilon \in P \text{ iff } q_i \in F}$$

\square

Claim 3. *A is the language generated by G .*

Proof. Every derivation in G is of the form

$q_0 \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \dots \Rightarrow a_1a_2\dots a_nq_n \Rightarrow a_1\dots a_n$ where $q_n \longrightarrow \varepsilon \in P$. By the construction of the grammar it follows that $\delta^*(q_0, a_1\dots a_n) = q_n \in F$

Similarly for every string $a_1\dots a_n \in \mathcal{L}(D)$ we have a collection of states $q_0, q_1, \dots, q_n \in Q$ with $q_n \in F$ such that $\delta^*(q_0, a_1\dots a_n) = q_n$. Correspondingly we have the derivation

$$q_0 \Rightarrow a_1q_1 \Rightarrow a_1a_2q_2 \Rightarrow \dots \Rightarrow a_1a_2\dots a_nq_n \Rightarrow a_1a_2\dots a_n. \quad \square$$

The two theorem proved above give us the following characterization

A language $L \subseteq \Sigma^*$ is regular iff there exists a right-linear grammar which generates L

Analogous to the notation of a right - linear grammar is the notion of a left-linear grammar in which all the productions are of the form $S \longrightarrow Tx$ where $T \in V \cup \{\varepsilon\}$ and $x \in \Sigma^*$. Both right-linear and left-linear grammars have the same power of generation. But to prove that left-linear grammars generate only regular languages we need the following lemma.

Lemma 6. *If $A \subseteq \Sigma^*$ is regular then so is A^R where $A^R = \{x^R | x \in \Sigma^*\}$ and for any string $x = a_1\dots a_m$. $x_R = a_m\dots a_1$ is the reverse of x .*

Proof. Since A is regular there exists a *NFA*

$$N = \langle Q, \Sigma, \Delta, S, F \rangle \text{ with } \mathcal{L}(N) = A.$$

Consider the *NFA*

$$N^R = \langle Q, \Sigma, \Delta^R, F, S \rangle$$

constructed from N by

- (i) making S the set of final states
- (ii) making F the set of start states
- (iii) taking the universe of the relation Δ as the transition relation.

$$\text{i.e } q \xrightarrow{a} q' \in \Delta \text{ iff } q' \xrightarrow{a} q \in \Delta^R$$

The result then follows from the following claim □

Claim 4. *For all $q_0 \in S$ and $q_f \in F$ and $x \in \Sigma^*$*

$$q_f \in \Delta^*(q_0, x) \text{ iff } q_0 \in \Delta^*(q_f, x^R)$$

Proof. By induction on the length of x . □

Corollary 1. *A language $A \subseteq \Sigma^*$ is regular iff A^R is regular iff $(A^R)^R$ is regular*

Proof. Follows from the fact that $A = (A^R)^R$.

We are now ready to prove our main theorem □

Theorem 10. *A language is regular iff there exists a left-linear grammar that accepts it.*

Proof. We prove that for every left-linear grammar there exists a corresponding right linear grammar which generates the reverse language and vice-versa.

Let \mathcal{G}_L be the class of left-linear grammars over Σ and \mathcal{G}_R the class of right-linear grammars. Define the following 1 – 1 correspondence.

$$f : \mathcal{G}_L \longrightarrow \mathcal{G}_R$$

such that

$$G_L = \langle V_L, \Sigma, S_L, P_L \rangle \quad \langle V_R, \Sigma, S_R, P_R \rangle$$

$$\begin{aligned}
 & f(G_L) = G_R \text{ iff} \\
 & V_L = V_R \quad , \quad S_L = S_R \text{ and for all } T \in V_L = V_R \\
 & \text{and } \cup \in V_L \cup \{\varepsilon\} = V_R \cup \{\varepsilon\} \text{ and } x \in \Sigma^* \\
 & T \longrightarrow x \cup \in P_R \text{ iff } T \longrightarrow \cup x^R \in P_R.
 \end{aligned}$$

Claim 5. $x \in \mathcal{L}(G_L)$ iff $x^R \in \mathcal{L}(G_L)$ for all $x \in \Sigma^*$ and G_R, G_L such that $G_R = f(G_L)$

Proof. By induction on the length of x .

Main proof

(\Rightarrow) If $A \subseteq \Sigma^*$ is regular then A^R is also regular and there exists G_R which generates A^R . From the claim above $f^{-1}(G_R) = G_L$ generates $(A^R)^R = A$. Hence every regular language may be generated by a left-linear grammar.

(\Leftarrow) Let G_L be any left-linear grammar. Then $G_R = f(G_L)$ is a right-linear grammar that generates $(\mathcal{L}(G_L))^R$. But then $(\mathcal{L}(G_L))^R$ must be regular which implies $\mathcal{L}(G_L)$ is also regular. \square

However we need to emphasize that a grammar in which there are both left-linear as well as right linear productions may generate a language that is not regular as the following example shows

Example $G = \langle \{S, A, B\}, \{a, b\}, S, P \rangle$ where $S \longrightarrow A$, $A \longrightarrow aB$, $A \longrightarrow \varepsilon$ and $B \longrightarrow Ab$ are the productions generates the language

$$\begin{aligned}
 S & \Rightarrow A \Rightarrow \varepsilon^{\{a^m b^m \mid m \geq 0\}} \\
 S & \Rightarrow \underbrace{A \Rightarrow aB \Rightarrow aAb \Rightarrow \dots \Rightarrow a^m Ab^m \Rightarrow a^m b^m}_{2m}
 \end{aligned}$$

The Pumping Lemma

Theorem 11. Let $A \subseteq \Sigma^*$ be a regular language. Then there exists $m > 0$ such that for any $x \in A$ with $|x| \geq m$ x may be decomposed into $x = uvw$ with $|uv| \leq m$ and $|v| > 0$ such that $x_i = uv^i w \in A$ for every $i \geq 0$.

$$\begin{aligned}
 & A \subseteq \Sigma^* \text{ is regular} \\
 \Rightarrow & \exists m > 0 : \forall x \in A : |x| \geq m \Rightarrow \\
 & \exists u, v, w : x = uvw \quad \wedge \quad |uv| \leq m \quad \wedge \quad |u| > 0 \\
 & \wedge \quad \forall k \geq 0 : x_k = uv^k w \in A.
 \end{aligned}$$

Proof. If A is finite then choose $m > \max\{|x| \mid x \in A\}$ and the theorem holds vacuously. So assume A is infinite. \square

Claim 6. If A is infinite then for each $n \geq 0$, there exists $x \in A$ such that $|x| > n$.

Proof. Suppose not. Then for some value of $n \geq 0$. every string in $x \in A$ has a length $\leq n$. But for any $k \geq 0$ and finite Σ , Σ^k is a finite set which implies $\bigcup_{k \leq n} \Sigma^k$ is also finite. In fact Σ^k has exactly $|\Sigma|^k$

different strings and $|\bigcup_{k \leq n} \Sigma^k| = \sum_{k \leq n} |\Sigma|^k$.

□

The contrapositive of the pumping lemma states that

$$\begin{aligned} \forall m > 0 : \exists x \in A : |x| \geq m \wedge \\ \forall u, v, w : x = uvw \wedge |uv| \leq m \wedge |v| > 0 \\ \Rightarrow \exists k \geq 0 : x_k = uv^k w \notin A \end{aligned}$$

$\Rightarrow A$ is not regular.

CHALLENGER'S STRATEGY

1. $x = a^m b^m$
2. $k = 0$

Example $A = \{a^n b^n | n \geq 0\}$ is not regular. Assume A is regular. Therefore the pumping lemma there must hold and A is infinite. Since A is regular there must be a *DFA* D which accepts A . Suppose the *DFA* has N states. Choose $m = N + 1$. Consider the string $a^m b^m$ and the sequence of states

$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} \dots \xrightarrow{a} q_N \xrightarrow{a} q_m \xrightarrow{b} q_{m+1} \xrightarrow{b} \dots \xrightarrow{b} q_{2m}$$

Clearly for some $i < j < N$, $q_i = q_j$

Choose $u = a^i, v = a^{j-i}, w = a^{m-j} b^m$. so that $x = a^i a^{j-i} a^{m-j} b^m = uvw$. Hence every $x_k = a^i (a^{j-i})^k a^{m-j} b^m \in A$. In particular $x_0 = a^i a^{m-j} b^m = a^{m-j+i} b^m \in A$. But since $j \neq i$, $m - j + i \neq m$. Hence $x_0 \notin A$ which is a contradiction. Hence the assumption that A is regular is wrong.

The proof proper Since A is regular there exists a *DFA* $D = \langle Q, \Sigma, \delta, q_0, F \rangle$ which accepts A . Let $|Q| = n$. Since A may be assumed to be infinite it does have strings of length $> n + 1$. Choose any such string $|x| \geq n + 1$. Then $\delta^*(q_0, x) = q_f \in F$ consider the sequence of states in the *DFA* which lead to q_f from q_0 . This is a sequence of $n + 2$ states.

$$q_0, q_1, \dots, q_{n-1}, q_n, q_f$$

By the pigeon hole principle there exists $i < j \leq n$ such that $q_i = q_j$. This implies that $x = uvw$ such that

$$\{\delta^*(q_0, u) = q_i \delta^*(q_i, v) = q_i = q_j \delta^*(q_j, w) = q_f\} \quad \begin{aligned} |uv| \leq n + 1 = m \\ |v| > 0 \end{aligned}$$

Assume $v = av'$ and $w = bw'$. Clearly $\delta(q_i, a) = q_{i+1}$ and $\delta(q_j, b) = q_{j+1} = \delta(q_i, b)$
It follows therefore that for any $k \geq 0$

$$\{\delta^*(q_0, u) = q_i \delta^*(q_i, v^k) = q_i \delta^*(q_i, w) = q_f\} \Rightarrow x_k = uv^k w \in A$$

The Pumping Lemma as a game

Let Σ be any alphabet ($\neq \phi$) and let $A \subseteq \Sigma^*$ be a language. The game is a 2-person game between DEFENDER who tries to defend the assertion “ A is regular” and a CHALLENGER who challenges the assertion. The game proceeds as follows :

1. DEFENDER : chooses a positive integer $m > 0$.
2. CHALLENGER : chooses an $x \in A$ with $|x| \geq m$.
3. DEFENDER : decomposes x into 3 parts u, v, w such that $v \neq \varepsilon$ and $|uv| \leq m$

4. CHALLENGER : chooses $k \geq 0$

Result : if $x_k = uv^k w \in A$ then DEFENDER wins.
else CHALLENGER wins.

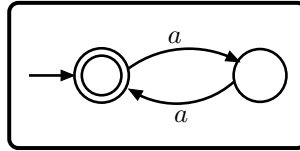
Now CHALLENGER has a winning strategy for $A \Rightarrow A$ is not regular.

However if DEFENDER has a winning strategy for A ?

MAIN QUESTION. ARE THERE NON-REGULAR LANGUAGES
FOR WHICH DEFENDER HAS A WINNING STRATEGY

Example. Determine whether $A = \{a^{2n} | n \geq 0\}$ is regular. Since the regular expression $(\mathbf{a a})^*$ is a representation of the language A the DEFENDER must have a winning strategy.

$(\mathbf{a a})^*$ is accepted by



D : choose $m = 2$. (no of states in the DFA).

C : choose any $n \geq 2$, $x = a^{2n}$

D : choose $i = 0$, $j = 2$ with $u = \varepsilon$ and $v = a^2 \Rightarrow x = a^{2n} = a^2 a^{2(n-1)} \Rightarrow w = a^{2(n-1)}$.

C : choose any $k \geq 0$.

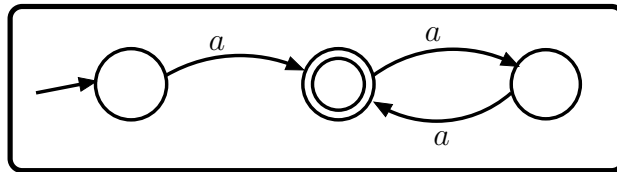
DEFENDER'S STRATEGY

1. $m = 3$

2. $u = \varepsilon, v = a^2$

Result : DEF has a winning strategy since for all $n \geq 2$ and all $k \geq 0$, $x_k = (a^2)^k a^{2(n-1)} = a^{2(n+k-1)} \in A$.

Example. Determine whether $A = \{a^{2n+1} | n \geq 0\}$ is regular.



D : choose $m = 3$.

C : choose any $n \geq 1$.

D : choose $i = 1$, $j = 2$ with $u = a$ and $v = a^2$
 $\Rightarrow x = uvw = a^3 a^{2(n-1)} \Rightarrow w = a^{2n-2} = a^{2(n-1)}$.

C : choose any $k \geq 0$.

$\Rightarrow x_k = a(a^2)^k a^{2(n-1)} = a^{2n-2+2k+1} = a^{2(n+k)+1} \in A$.

DEFENDER'S STRATEGY

1. $m = 3$
2. $i = 1, j = 2$ i.e. $u = a, v = a^2$

Example Determine whether $A = \{a^{n^2} | n \geq 0\}$ is regular. We prove that C has a winning strategy.

D : Consider any $m > 0$.

C : Consider any $n > m > 0$. $\Rightarrow n^2 > n > m > 0, x = a^{n^2}$

D : Chooses any $i \geq 0, j > 1$ such that $i + j \leq m$.

$\Rightarrow 1 < j \leq m < n < n^2$.

$\Rightarrow a^i v = a^i w = a^{(n^2-(i+j))}$.

C : Challenger chooses k such that

$$x_k = a^i (a^j)^k a^{n^2-(i+j)} = a^{n^2+kj} \notin A.$$

CHALLENGER'S STRATEGY

1. $x = a^{n^2}, n^2 > n > m > 0$
2. $k = 1$

challenger has a winning strategy provided.

$n^2 + k_j$ is not a perfect square for C 's choice of k . Since $1 < j < n$ we have

$$n^2 < n^2 + j < n^2 + n < n^2 + 2n + 1 = (n + 1)^2$$

Hence challenger chooses $k = 1$ and it is guaranteed that $n^2 < n^2 + j < (n + 1)^2$ and hence $n^2 + j$ is never a perfect square.

Example Consider the language $A = \{a^{2^n} | n \geq 0\}$.

D : Chooses some $m > 0$.

C : Chooses $x = a^{2^n}$ with $n \gg m > 0$

D : Chooses $i \geq 0, j > 0$ with $u = a^i, v = a^j$
such that $i + j \leq m$.

$\Rightarrow x = u v w$ and $w = a^{(2^n-(i+j))}$

C : Chooses $k = 2 \Rightarrow x_k = a^i a^{2j} a^{(2^n-(i+j))} = a^{2^n + j}$

CHALLENGER'S STRATEGY

1. $x = a^{2^n}, n > m > 0$
2. $k = 2$

Result : C always wins provided $2^n + j$ is not a power of 2. $2^n + j$ is a power of 2 for $n > 0$ only when $j = 2^n$. But $n > m > 0 \Rightarrow 2^n > m$. But $j \leq m \Rightarrow j = 2^n$ is impossible.

Example of a non-regular language which satisfies the condition of the pumping lemma

Let $A = \mathcal{L}(\mathbf{b}^* \mathbf{c}^*) = \{b^m c^n | m, n \geq 0\}$

$B = \{ab^k c^k | k \geq 0\}$ (\star this is clearly a context-free language)

$$C = \{a^{i+2}b^j c^k | i, j, k \geq 0\} = \mathcal{L}(\mathbf{aa a^*b^*c^*})$$

Claim 7. $A \cup B \cup C$ is not regular.

Proof. Note : $B \not\subseteq A \cup C$ since $\forall x \in A \cup C : \#a(x) \neq 1$.
whereas $\forall y \in B : \#a(y) = 1$.

In fact A, B, C are mutually disjoint .

Proof by contradiction : Assume $L = A \cup B \cup C$ is regular and accepted by a DFA $D = \langle Q, \{a, b, c\}, \delta, q_0, F \rangle$.
Since $A \cap B = B \cap C = C \cap A = \phi$ we have $B = L - (A \cup C) = L \cap \sim (A \cup B)$ from closure properties that B is regular which is a contradiction. \square

Claim 8. $|Q| > 1$.

Proof. Since $q_0 \in Q, |Q| \geq 1$. Since $a \in B, \delta(q_0, a) \neq q_0$ otherwise any a^n for $n \geq 0$ would be accepted by D which is not the case. Hence $|Q| > 1$.

Assume $|Q| = n > 1$. Consider any string $x = ab^k c^k \in B$, such that $k > n$. Let

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} \dots \xrightarrow{b} q_k \xrightarrow{b} q_{k+1} \xrightarrow{c} \dots \xrightarrow{c} q_{2k+1}$$

Since $k > n$, there must be states $1 \leq i \leq j \leq k$ such that $a_i = q_j$.

This implies that

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} \dots \xrightarrow{b} q_i = \underbrace{q_j \xrightarrow{b} \dots \xrightarrow{b} q_j}_{\text{a cycle which may be pumped.}} \xrightarrow{b} \dots \xrightarrow{b} q_{k+1} \xrightarrow{c} \dots \xrightarrow{c} q_{2k+1}$$

$x = uvw$ where $u = ab^{i-1}$, $v = b^{j-i}$, $w = b^{k+1-j}c^k$ and hence for any $x_0 = uw$ is accepted by D .

But $uw = ab^{i-1}b^{k+1-j}c^k$	But $1 \leq i < j \leq k$ $\Rightarrow j - i \Rightarrow -(j - i) < 0$ $\Rightarrow k - (j - i) < k$
$= ab^{k-(j-i)}c^k$	
$\notin B$ since $k - (j - i) \neq k$	
$A \cup C$ since $\#a(uw) = 1$.	

Hence x_0 is accepted by D but $x_0 \notin A \cup B \cup C$. Hence there is no DFA which accepts $A \cup B \cup C$

DEFENDER'S WINNING STRATEGY

1. $m = 2$
2. $u = \varepsilon$ and $v = \begin{cases} a^2 & \text{if } x \in \mathbf{a^2b^*c^*} \\ x(1) & \text{else} \end{cases}$

\square

Claim 9. By choosing $m = 2$, DEFENDER has a winning strategy.

Proof. Let CHALLENGER choose any $x \in A \cup B \cup C$ with $|x| \geq 2$
The DEFENDER then has the following cases

Case 0. $x \in A$ with $|x| \geq 2$

$$\Rightarrow x = b^i c^j \text{ for some } i, j \geq 0 \text{ and } i + j \geq 2$$

$$\Rightarrow x = \begin{cases} c^i & \text{with } j \geq 2 \text{ and } i = 0 \\ b^i & \text{with } i \geq 2 \\ b^i c^j & \text{with } i \geq 1, j \geq 1 \end{cases}$$

DEFENDER chooses $u = \varepsilon$ and $v = x(1)$

$$\Rightarrow x = \begin{cases} b^i \text{ with } i \geq 2 \Rightarrow v = b \wedge w = b^{i-1} \Rightarrow v^k w = b^{k+i-1} \in A \\ c^j \text{ with } j \geq 2 \Rightarrow v = c \wedge w = c^{j-1} \Rightarrow v^k w = c^{k+j-1} \in A \\ b^i c^j \text{ with } i \geq 1, j \geq 1 \Rightarrow v = b \wedge w = b^{i-1} c^j \Rightarrow v^k w = b^{k+i-1} c^j \in A \end{cases}$$

In each case $v^k w \in A$, for all $k \geq 0$ and hence DEFENDER wins.

Case 1. $x \in B$ with $|x| \geq 2$

$\Rightarrow x = a b^i c^j$ with $i \geq 1$. (if $i = 0$ then $|x| < 2$). DEFENDER chooses $u = \varepsilon$ and $v = x(1) = a$. Then for every choice of k , we have

$$v^k w = \begin{cases} b^i c^j \in A & \text{if } k = 0 \\ a b^i c^j \in B & \text{if } k = 1 \\ a^k b^i c^j \in C & \text{if } k > 1 \end{cases}$$

and the DEFENDER wins.

Case 2. $x \in C$ with $|x| \geq 2$

$$\begin{aligned} \Rightarrow x &= a^{2+i} b^j c^l \text{ with } i, j, l \geq 0. \\ \Rightarrow x &= \begin{cases} a^2 b^j c^l & \text{with } i = 0, l \geq 0. \\ a^{i+3} b^j c^l & \text{with } i \geq 0, j, l \geq 0. \end{cases} \end{aligned}$$

Case 2.2 $x = a^2 b^j c^l$

Then DEFENDER chooses $u = \varepsilon$ and $v = a^2$

$$\Rightarrow w = b^j c^l \text{ with } j, l \geq 0$$

for any $k \geq 0$ we have $v^k w = a^{2k} b^j c^l$

$k = 0 \Rightarrow v^k w = b^j c^l \in A$. and DEFENDER wins

$k > 0 \Rightarrow v^k w = a^{2k} b^j c^l \in A$. and DEFENDER wins

Case 2.3 $x = a^{i+3} b^j c^l$ with $i \geq 0, j, l \geq 0$

The DEFENDER chooses $u = \varepsilon$ and $v = a$

$$\Rightarrow w = a^{i+2} b^j c^l$$

and for all $k \geq 0$, $v^k w = a^{i+k+2} b^j c^l \in C$ and DEFENDER wins.

DEFENDER'S STRATEGY (summary)

1. $m = 2$
2. For any $x \in A \cup B \cup C$,

$$u = \varepsilon \quad \text{and} \quad v = \begin{cases} a^2 & \text{if } x \in \mathbf{a^2 b^* c^*} \\ x(1) & \text{otherwise} \end{cases}$$

□

Other Application of Pumping Lemma

Theorem 12. Let D be a DFA with m states. Then $\mathcal{L}(D) \neq \phi$ iff there exists a string $x \in \mathcal{L}(D)$ such that $|x| < m$

Proof. (\Rightarrow) Let x be a minimal length string in $\mathcal{L}(D)$. If $|x| \geq m$ then by the pumping lemma $x = uvw$ with $|uv| \leq m$ and $v \neq \varepsilon$. Clearly $uv^0w = uv^0w \in \mathcal{L}(D)$. But since $|v| > 0$ $|uv^0w| < |x|$ contradicting the assumption that x is a minimal length string in $\mathcal{L}(D)$. □

The above theorem finishes a decision procedure to test whether a given DFA, D accepts the empty language. Since the number of strings of length $< m$ is finite, it is necessary to run the DFA on all possible strings of length $< m$, to determine it.

Theorem 13. *If A is an infinite regular language then there exist strings u, v, w such that $uv^i w \in A$ for all $i \geq 0$.*

Proof. Obvious □

Theorem 14. *Let D be a DFA with m states $m > 0$. Then $\mathcal{L}(D)$ is infinite iff $\mathcal{L}(D)$ contains a string x , with $m \leq |x| < 2m$.*

Proof. (\Leftarrow) Assume $x \in \mathcal{L}(D)$ with $m \leq |x| < 2m$. By the pumping lemma x may be decomposed into u, v, w with $v \neq \varepsilon$ and such that for all $i \geq 0$, $uv^i w \in \mathcal{L}(D)$. Hence $\mathcal{L}(D)$ is infinite.

(\Rightarrow) Assume $\mathcal{L}(D)$ is infinite. Then there exist strings $x \in \mathcal{L}(D)$ with $|x| \geq 2m$. Let x be the shortest such string and $x = x_1 x_2$ where $|x_1| = m$ and $|x_2| \geq m$. Consider the proof of the pumping lemma with $q = \delta(q_0, x_1)$. Clearly the path from q_0 to q touches $(m + 1)$ states while accepting x_1 . This implies there exists a q' which appears twice

$$q_0 \longrightarrow \dots \longrightarrow q' \longrightarrow \dots \longrightarrow q' \longrightarrow \dots \longrightarrow q.$$

Hence x_1 may be decomposed into u, v, w such that $v \neq \varepsilon$, $x_1 = uvw$ and $\delta(q_0, u) = q'$, $\delta(q', v) = q'$ and $\delta(q', w) = q$. Clearly $x = uvwx_2 \in \mathcal{L}(D)$ and state q' is repeated implies $x' = uvwx_2 \in \mathcal{L}(D)$. Further since D is a DFA, $v \neq \varepsilon$. Hence $|\varepsilon| > 0$, $|v| \leq m$.

$$|x'| = |uvwx_2| \geq |x_2| \geq m. \text{ and since } v \neq \varepsilon, |x'| < |x|$$

But since x is the shortest word of length $\geq 2m$. It follows that $|x'| \not\geq 2m$ and must be $< 2m$. But $|x'| \geq m$. Hence $m \leq |x'| < 2m$ and $x' \in \mathcal{L}(D)$ □

Decision procedure to determine whether the language accepted by a DFA is finite

Simply run the DFA on all strings of length in the interval $(m, 2m)$. If it accepted none then the DFA accepts only a finite language. Otherwise it accepts an infinite language.

Ultimate Periodicity

Definition 11. *A subset $\cup \subseteq \mathbb{N} = \{0, 1, 2, \dots\}$ is said to be ultimately periodic if there exist $m \geq 0$, $p > 0$ such that for all $n \leq m$, $n \in \cup$ iff $n + p \in \cup$.*

Example

1. Every finite subset \cup of \mathbb{N} is ultimately periodic with $m = \max \cup + 1$ and any $p > 0$.
2. Every infinite arithmetic sequence is ultimately periodic with a period defined by the common difference between successive numbers and m the starting number
3. The union of any finite collection of infinite arithmetic sequences with different starting points but the same common difference is also ultimately periodic. That is if $A_i = \{m_i + j^d | j \geq 0, m_i \geq 0\}$, $i > 0$ is a finite collection of $k \geq 0$ infinite sets. Then $\cup = \bigcup_{i \leq k} A_i$ is ultimately periodic with $m = \max m_i$ and period d .

HOMOMORPHISMS

Definition 12. Let Σ and Γ be alphabets. Then any function $h : \Sigma^* \rightarrow \Gamma^*$ is a homomorphism if $\forall x, y \in \Sigma^* : h(xy) = h(x)h(y)$

Definition 13. Let $h_0 : \Sigma \rightarrow \Gamma^*$. $h : \Sigma^* \rightarrow \Gamma^*$ is said to be a homomorphic extension of h_0 if
(i) h is a homomorphism
(ii) $\forall a \in \Sigma : h(a) = h_0(a)$.

Lemma 7. For any homomorphism $h : \Sigma^* \rightarrow \Gamma^*$, $h(\varepsilon) = \varepsilon$

Proof. Assume not. Then $h(\varepsilon) = u \neq \varepsilon$. This implies for any $x \in \Sigma^*$
 $v = h(x) = h(\varepsilon x) = h(\varepsilon)h(x) = uv = u^2v = \dots$
which implies h is not even a function from Σ^* to Γ^* □

Lemma 8. Every $h_0 : \Sigma \rightarrow \Gamma^*$ has a unique homomorphic extension.

Proof. Consider $h : \Sigma^* \rightarrow \Gamma^*$ a homomorphic extension of h_0 . Since h is a homomorphism we have

$$h(\varepsilon) = \varepsilon$$

and $\forall a \in \Sigma : h(a) = h_0(a)$. and $\forall x, y \in \Sigma^* : h(xy) = h(x)h(y)$. If h' is any other homomorphic extension of h_0 . We have

$$h'(\varepsilon) = \varepsilon = h(\varepsilon)$$

and by induction hypothesis is (assuming $h'(x) = h(x)$ for all $|x| < n$) if $y = ax$ then $h'(y) = h'(ax) = h_0(a)h'(x) = h_0(a)h(x) = h(ax)$. Hence $h = h'$ and homomorphic extensions are unique. □

Definition 14. Let $h : \Sigma^* \rightarrow \Gamma^*$ be a homomorphism. For any $A \subseteq \Sigma^*$ $h(A) = \{h(x) | x \in A\}$ and for any $B \subseteq \Gamma^*$ $h^{-1}(B) = \{x \in \Sigma^* | h(x) \in B\}$. $h(A)$ is called the image of A under h and $h^{-1}(B)$ is called the pre-image of B under h .

Theorem 15. Let $h : \Sigma^* \rightarrow \Gamma^*$ be a homomorphism. for any regular $B \subseteq \Gamma^*$, $h^{-1}(B)$ is also regular.

Proof. Let $D_B = \langle Q_B, \Gamma, \delta_B, q_0, F_B \rangle$ be a DFA that accepts B . Let $A = h^{-1}(B)$. Construct the DFA $D_A = \langle Q_A, \Sigma, \delta_A, q_0, F_A \rangle$ with $Q_A = Q_B$, $F_A = F_B$ and the same start state q_0 . Define $\delta_A(q, a) = \delta_B^*(q, h(a))$ for all $q \in Q_A$, $a \in \Sigma$. □

Claim 10. $\delta_A^*(q, x) = \delta_B^*(q, h(x))$ for all $x \in \Sigma$

Proof. By induction on $|x|$ since $\delta_A(q, \varepsilon) = q = \delta_B(q, \varepsilon)$ □

Claim 11. $\mathcal{L}(D_A) = h^{-1}(\mathcal{L}(D_B))$.

Proof. $x \in \mathcal{L}(D_A) \Leftrightarrow \delta_A^*(q_0, x) \in F$
 $\Leftrightarrow \delta_A^*(q_0, h(x)) \in F$
 $\Leftrightarrow h(x) \in \mathcal{L}(D_B)$
 $\Leftrightarrow x \in h^{-1}(\mathcal{L}(D_B))$ □

Theorem 16. Let $h : \Sigma^* \rightarrow \Gamma^*$ be a homomorphism. If $A \subseteq \Sigma^*$ is regular then so is $h(A)$.

Proof. If $A \subseteq \Sigma^*$ is regular then there exists a regular expression $\alpha \in RE_\Sigma$ such that $\mathcal{L}(\alpha) = A$. Let $h(A) = B \subseteq \Gamma^*$. We define a function

$$RE_\Sigma \rightarrow RE_\Gamma$$

which translates regular expression α into β i.e. $\mathfrak{h}(\alpha) = \beta$. We define \mathfrak{h} by induction on the structure of α as follows.

$$\begin{aligned} \mathfrak{h}(\phi) &= \phi \\ \mathfrak{h}(\varepsilon) &= \varepsilon \end{aligned}$$

$$\begin{aligned}
\mathfrak{h}(\emptyset) &= \text{if } h(a) = y, \text{ for each } a \in \Sigma. \\
\mathfrak{h}(\alpha + \alpha') &= \mathfrak{h}(\alpha) + \mathfrak{h}(\alpha') \\
\mathfrak{h}(\alpha.\alpha') &= \mathfrak{h}(\alpha).\mathfrak{h}(\alpha') \\
\mathfrak{h}(\alpha^*) &= (\mathfrak{h}(\alpha))^*
\end{aligned}$$

□

Claim 12. For any $\alpha \in RE_\Sigma$, $\mathcal{L}(\mathfrak{h}(\alpha)) = h(\mathfrak{h}(\alpha))$.

Proof. Follows by induction on the structure of α

□

Theorem 17. Any set $A \subseteq \{a\}^*$ is regular iff the set $\{m|a^m \in A, m \leq 0\}$ is ultimately periodic.

Proof. (\Rightarrow) If A is finite then take $m = \max\{i|a^i \in A\} + 1$. Otherwise let A be infinite and consider any DFA D which accepts A . Clearly since it is deterministic the transition graph of the DFA (restricted to the accessible states) looks as follows where there is an initial

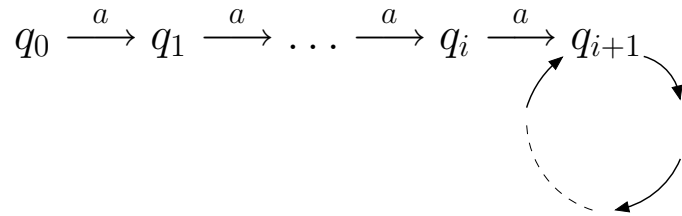


Figure 1:

linear sequence of distinct states followed by a loop of length $p \geq 1$

Choose $m =$ the number of distinct initial states and p the length of the loop.

Note If the path through the DFA extend beyond the loop

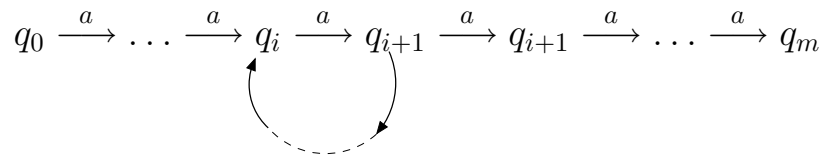


Figure 2:

then an equivalent DFA which accepts the same language may be constructed which conforms to Fig 1. (\Leftarrow) Conversely let \cup be any ultimately periodic set with period $p > 0$ and starting point n_4 . Then construct a $DFA_{\{a\}}$ with $n + p + 1$ states which accepts exactly the set $\{a^m|m \in \cup\}$ □

Application of homomorphisms

Corollary 2. Let $A \subseteq \Sigma^*$ be any regular language. Then the set $\text{sup} = \{|x| \mid x \in A\}$ is ultimately periodic.

Proof. Since A is regular, $h(A)$ is also regular where $h : \Sigma \rightarrow \{1\}$ is defined as $\forall a \in \Sigma : h(a) = 1$. From the previous theorem it follows that $h(A)$ is ultimately periodic. But $h(A)$ is merely the unary representation of the elements of \cup .

Example

$A = \{a^i b^j c^{i+j} \mid i \geq 0, j \geq 0\}$ can be proven to be non-regular by considering the homomorphism $h(a) = h(b) = b$ and $h(c) = c$. Then $h(A) = \{b^{i+j} c^{i+j} \mid i + j \geq 0\}$ which is clearly non-regular. \square

DFA MINIMIZATION

In general, for any regular language there may be several possible DFA designs. It is possible to minimize the number of states in two stages :

- (i) Get rid of inaccessible states. A state is inaccessible if there is no path to it from the start state.
- (ii) Collapse states which are “equivalent” in some sense without changing the language accepted.

In general inaccessible states may be identified by “walking” forwards from the start state to the final states and marking all the the states reached. Hence a depth-first search algorithm or a breadth-first search algorithm should do the job.

In more interesting question is that of collapsing equivalent states.

Given a DFA^D consisting only of accessible states, for any string x , $x \in \mathcal{L}(D)$ iff $\delta^*(q_0, x) \in F$. If $q = \delta^*(q_0, x) \notin F$ then q is clearly a reject state.

Definition 15. Two states p, q are said to be indistinguishable (denoted $p \approx q$) iff $\forall x \in \Sigma^* : \delta^*(q_0, x) \in F \Leftrightarrow \delta^*(q, x) \in F$. p is said to be distinguishable from q if there exists a distinguished word $w \in \Sigma^*$ such that either $\delta^*(p, w) \in F \wedge \delta^*(q, w) \notin F$ or $\delta^*(p, w) \notin F$ and $\delta^*(q, w) \in F$.

Fact

1. Indistinguishability is an equivalence relation on the states of a DFA.

i.e. $\forall p \in Q : p \approx p$ reflexivity
 $\forall p, q \in Q : p \approx q \Rightarrow q \approx p$ symmetry
 $\forall p, q, r \in Q : p \approx q \wedge q \approx r \Rightarrow p \approx r$ transitivity.

2. \approx partitions the set of states Q into equivalence classes.

$$\left. \begin{array}{l} [p]_{\approx} \stackrel{df}{=} \{q \in Q \mid p \approx q\} \\ p \approx q \Leftrightarrow [p]_{\approx} = [q]_{\approx} \end{array} \right\} \begin{array}{l} \text{for any } p, q \in Q \\ [p]_{\approx} \cap [q]_{\approx} = \phi \\ \vee [p]_{\approx} = [q]_{\approx} \end{array}$$

$$Q / \approx \stackrel{df}{=} \{ [p] \mid p \in Q \}.$$

3. For any DFA $D = \langle Q, \Sigma, \delta, q_0, F \rangle$,

$$D / \approx = \langle Q / \approx, \Sigma, \delta_{\approx}, [q_0]_{\approx}, F \approx \rangle$$

where $\delta_{\approx}([p], a) = [\delta(p, a)]_{\approx}$

We now have to show that this quotient construction is well-defined and that D / \approx is indeed a DFA. This involves showing that δ_{\approx} well-defined.

Lemma 9. If $p \approx q$ then $\delta(p, a) \approx \delta(q, a)$ for all $a \in \Sigma$

Proof. $p \approx q \Leftrightarrow \forall x \in \Sigma^* : \delta^*(p, x) \in F \Leftrightarrow \delta^*(q, x) \in F$.

Consider any $a \in \Sigma$ and $y \in \Sigma^*$.

Let $\delta(p, a) = p_a$ and $\delta(q, a) = q_a$. For any $y \in \Sigma^*$

we have

$$\left. \begin{array}{l} \delta^*(p_a, y) \in F \\ \Leftrightarrow \delta^*(p, ay) \in F \\ \Leftrightarrow \delta^*(q, ay) \in F \\ \Leftrightarrow \delta^*(q_a, y) \in F \end{array} \right\} \Rightarrow p_a \approx q_a$$

□

form this lemma it follows that the definition

$$\delta_{\approx}([p]_{\approx}, a) = [\delta(p, a)]_{\approx} \text{ is well defined.}$$

Lemma 10. $p \in F \Leftrightarrow [p]_{\approx} \in F/\approx$

Proof. (\Rightarrow) follows from the definition.

(\Leftarrow) Let $q \in [p]_{\approx} \in F/\approx$.

Clearly if $p \in F$ then $\delta^*(p, \varepsilon) = p \in F$ and since $q \approx p$, $\delta^*(q, \varepsilon) = q \in F$.

□

Lemma 11. $\forall x \in \Sigma^* : \delta_{\approx}^*([p]_{\approx}, x) = [\delta^*(p, x)]_{\approx}$.

Proof. By induction on $|x|$.

We now prove that D/\approx accepts the same language as D .

□

Theorem 18. $\mathcal{L}(D/\approx) = \mathcal{L}(D)$

Proof. For any $x \in \Sigma^*$, $x \in \mathcal{L}(D/\approx)$

$$\Leftrightarrow \delta_{\approx}^*([q_0]_x, x) \in F/\approx$$

$$\Leftrightarrow [\delta^*(q_0, x)]_{\approx} \in F/\approx$$

$$\Leftrightarrow \delta^*(q_0, x) \in F$$

$$\Leftrightarrow x \in \mathcal{L}(D)$$

□

Consider the indistinguishability relation on the states of D/\approx . That is let

$$[p]_{\approx} \sim [q]_{\approx} \Leftrightarrow \forall x \in \Sigma^* : \delta_{\approx}^*([p]_{\approx}, x) \in F/\approx \Leftrightarrow \delta_{\approx}^*([q]_{\approx}, x) \in F/\approx$$

Claim 13. $[p]_{\approx} \sim [q]_{\approx} \Rightarrow [p]_{\approx} = [q]_{\approx}$.

Proof. $[p]_{\approx} \sim [q]_{\approx}$

$$\Rightarrow \forall x \in \Sigma^* : \delta_{\approx}^*([p]_{\approx}, x) \in F/\approx \Leftrightarrow \delta_{\approx}^*([q]_{\approx}, x) \in F/\approx$$

$$\Rightarrow \forall x \in \Sigma^* : [\delta^*([p], x)]_{\approx} \in F/\approx \Leftrightarrow [\delta^*([q], x)]_{\approx} \in F/\approx$$

$$\Rightarrow \forall x \in \Sigma^* : \delta^*(p, x) \in F \Leftrightarrow \delta^*(q, x) \in F$$

$$\Rightarrow p \approx q$$

$$\Rightarrow [p]_{\approx} = [q]_{\approx}$$

□

CONTEXT-FREE LANGUAGES

Definition 16. A grammar $G = \langle V, \Sigma, S, P \rangle$ is linear if every production has at most one variable on the right hand side.

Example The following grammar G is linear but neither right linear nor left-linear.

$$\begin{array}{l} S \longrightarrow A \\ A \longrightarrow aB|\varepsilon \\ B \longrightarrow Ab \end{array} \quad \parallel \quad \begin{array}{l} \text{Equivalently} \\ S \longrightarrow \varepsilon|aB \\ B \longrightarrow Sb \end{array}$$

$\mathcal{L}(G) = \{a^n b^n \mid n \geq 0\}$.

Definition 17. A grammar $G = \langle V, \Sigma, S, P \rangle$ is called context-free if all productions have the form $A \longrightarrow \alpha$ where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$. The language generated $\mathcal{L}(G)$ by a context-free grammar is called a context-free language.

Example $S_{>} \longrightarrow AS_{=} , S_{=} \longrightarrow aS_{=} b \mid \varepsilon , A \longrightarrow aA \mid a$ is a grammar with start symbol $S_{>}$ that generates $\{a^m b^n \mid m > n \geq 0\}$. Similarly

$$S_{<} \longrightarrow S_{=} B , B \longrightarrow bB \mid b$$

generate $\{a^m b^n \mid 0 \leq n < m\}$. Then $S_{\neq} \longrightarrow S_{>} \mid S_{<}$ generates the language $\{a^m b^n \mid m \neq n\}$.

Example $S \longrightarrow aSb \mid SS \mid \varepsilon$ is a grammar that generates balanced parentheses sequence. This grammar is context-free but is not linear. The language is described by

$$\{w \in \{a, b\}^* \mid \#a(w) = \#b(w) \wedge \forall v \preceq w : \#a(v) \geq \#b(v)\}$$

Facts And Remarks

F1. Every right linear or left linear language is a linear language.

F2. Every linear language is context-free.

R3. Whereas the language $\{a^m b^n \mid m \neq n\}$ has been generated by a non linear CF grammar, there does exist a linear grammar which can generate it.

R4. The example above is of a language that cannot be generated by a linear grammar.

Derivation Trees

Let $G = \langle V, \Sigma, S, P \rangle$ be a CFG. A rooted ordered tree is called a derivation tree for G iff it satisfies the following conditions.

0. The root is labelled S .
1. Every leaf has a unique label from $\Sigma \cup \{\varepsilon\}$.
2. Any leaf labelled ε has no other siblings
3. Every non-leaf node is labelled (uniquely) by a symbol from V .
4. If a non-leaf node has a label A and its children from left to right are labelled respectively with symbols that form a string $al \in (V \cup \Sigma)^+$ then $A \longrightarrow \alpha \in P$.

A partial derivation tree is one which is like a derivation tree except that the condition 1 is replaced by

- 1'. Every leaf has a unique label from $V \cup \Sigma \cup \{\varepsilon\}$.

Definition 18. The yield of a tree is the string of symbols obtained by reading the labels on the leaves from left to right (omitting all occurrences of ε).

fact Every derivation tree is also a partial derivation tree

Proof that the set of all balanced parenthesis is generated by the grammar

$$S \longrightarrow aSb \mid SS \mid \varepsilon$$

The set $BP = \{w \in \{a, b\}^* \mid \#a(w) = \#b(w) \wedge \forall v \preceq w : \#a(v) \geq \#b(v)\}$

The proof is divided into two claims for the

Lemma 12. $S \Rightarrow^* x \in \{a, b\}^*$ implies $x \in BP$.

Proof. By induction on the length of the derivation $S \Rightarrow^* x$. We actually prove by induction on the length of derivations for all sentential forms generated by the grammar that

$$S \Rightarrow^* \alpha \in (V \cup \Sigma \cup \{\varepsilon\})^* \text{ implies}$$

$$\alpha \in BP' = \{\beta \in (V \cup \Sigma \cup \{\varepsilon\})^* \mid \#a(\beta) = \#b(\beta) \wedge \forall \alpha' \preceq \beta : \#a(\alpha') \geq \#b(\alpha')\}$$

By induction on the length of the derivation $S \Rightarrow^* \alpha$.

Basis 0. $\alpha = S$ and $S \in BP'$.

Ind Step. Assume for all derivations of length $\leq k$, $S \Rightarrow^* \alpha$, $\alpha \in BP'$. Consider any derivation of length $k+1$ with $S \Rightarrow^* \alpha \Rightarrow \beta$.

By IH $\alpha \in BP'$. By case analysis we have the following possibilities.

$$\left. \begin{array}{l} (i) \alpha = \alpha_1 S \alpha_2 \Rightarrow \alpha_1 a S b \alpha_2 = \beta \\ (ii) \alpha = \alpha_1 S \alpha_2 \Rightarrow \alpha_1 S S \alpha_2 = \beta \\ (iii) \alpha = \alpha_1 S \alpha_2 \Rightarrow \alpha_1 \alpha_2 = \beta \end{array} \right\} \text{ In each case it is clear that } \beta \in BP'$$

□

Lemma 13. Every $x \in BP$ is generated by the grammar.

Proof. By the induction on the length of x .

Basis $|x| = 0 \Rightarrow x = \varepsilon$ and $S \Rightarrow \varepsilon$ by the rule $S \longrightarrow \varepsilon$.

Induction Step If $|x| > 0$ then we have two cases.

$$\begin{array}{l} \text{case(i)} \quad \exists y \prec: y \in BP \\ \text{case(ii)} \quad \neg \exists y \prec: y \in BP \end{array}$$

$$\text{case(i)} \quad \exists y \prec: y \in BP \Rightarrow x = yz \text{ for some } z.$$

Claim 14. $z \in BP$

Proof. Since $\#a(x) = \#b(x)$ and $\#a(y) = \#b(y)$
 $\#a(x) = \#a(y) + \#a(z)$ and $\#b(x) = \#b(y) + \#b(z)$
we have $\#a(z) = \#a(x) - \#a(y)$
 $= \#b(x) - \#b(y)$

$$= \#b(z)$$

For any prefix $w \preceq z$ we have

$$\begin{aligned} \#a(w) &= \#a(yw) - \#a(y) \\ &\geq \#(yw) - \#b(y) \\ &= \#b(w) \end{aligned}$$

□

Hence since $y \in BP$ and $z \in BP$ and $|y| < |x|$ and $|z| < |x|$ we have by IH

$$\begin{aligned} S &\Rightarrow^* y \\ S &\Rightarrow^* z. \end{aligned}$$

It follows that the following derivation generate x

$$S \Rightarrow SS \Rightarrow^* yS \Rightarrow^* yz.$$

case(ii) $\neg \exists y \prec: y \in BP$ and $|x| > 0$.

This implies $x = azb$ for some $z \in BP$. and $|z| < |x|$. Hence there exists a derivation $S \Rightarrow^* z$. Therefore the following derivation generate x

$$S \Rightarrow aSb \Rightarrow^* azb = x$$

□

Definition 19. A leftmost derivation is a derivation in which the leftmost variable is rewritten. Similarly we have rightmost derivations.

Fact. For every derivation tree there is a unique leftmost (rightmost) derivation which corresponds to a depthfirst traversal of the tree in which the leftmost rightmost subtree is explored before any other subtree.

Theorem 19. Let $G = \langle V, \Sigma, S, P \rangle$ be a CFG. Then for every $w \in \mathcal{L}(G)$ there exists a derivation tree whose yield is w .

Proof. Since every derivation tree DT is also a partial derivation tree PDT it suffices to prove the following claim. □

Claim 15. For every sentential form α of G there exists a partial derivation tree whose yield is α .

Proof. By induction on the number of steps in a derivation of G that ends in α .

Basis S is the only sentential form derivable in 0 steps.

Induction Step Assume for every sentential form α derivable in k steps there exists a PDT of depth k . Let $\alpha = \beta$ in one step. Hence $\alpha = \alpha_1 A \alpha_2$ for some $\alpha_1, \alpha_2 \in (V \cup \Sigma)^*$ and $\beta = \alpha_1 \gamma \alpha_2$ for some $A \rightarrow \gamma \in P$. Consider the PDT for α , which exists by IH. The yield of this PDT is α . Hence there is a leaf node A preceded by leaf nodes of the symbols of α_1 and succeeded by the symbols of α_2 . By expanding the node labelled A to leaf nodes corresponding to the order of symbols in γ we get a new PDT whose yield is $\alpha_1 \gamma \alpha_2 = \beta$. □

Theorem 20. Every PDT of G represents some sentential form of G .

Proof. By induction on the maximum depth of *PDTs*.

Basis 0. The only sentential form is the single root node labelled S which is also the leaf node.

Induction Steps. Assume every *PDT* of depth $\leq k > 0$ represents a sentential form α of G where the yield of the *PDT* is α .

Consider any *PDT* of depth $k + 1$. Consider all the non-leaf nodes at depth k . Clearly all of them are variable symbols say $A_1, \dots, A_j, j > 0$. For each $A_i, 1 \leq i \leq j$ consider the ordered sequence of children $\alpha_1, \dots, \alpha_j$. Clearly there are productions $A_i \rightarrow \gamma_i \in P$ which yielded this tree. Consider the yield of the *PDT* upto depth k . By the *IH* it represents a sentential form.

$$\alpha = x_0 A_1 x_1 A_2 \dots A_j x_j$$

where $x_0, \dots, x_j \in \Sigma^*$. Again this is a sentential form of the grammar there must be a derivation

$$S \Rightarrow^* \alpha = x_0 A_1 x_1 \dots A_j x_j$$

Also the yield of the *PDT* of depth $k + 1$ is of the form

$$\beta = x_0 \gamma_1 x_1 \dots \gamma_j x_j$$

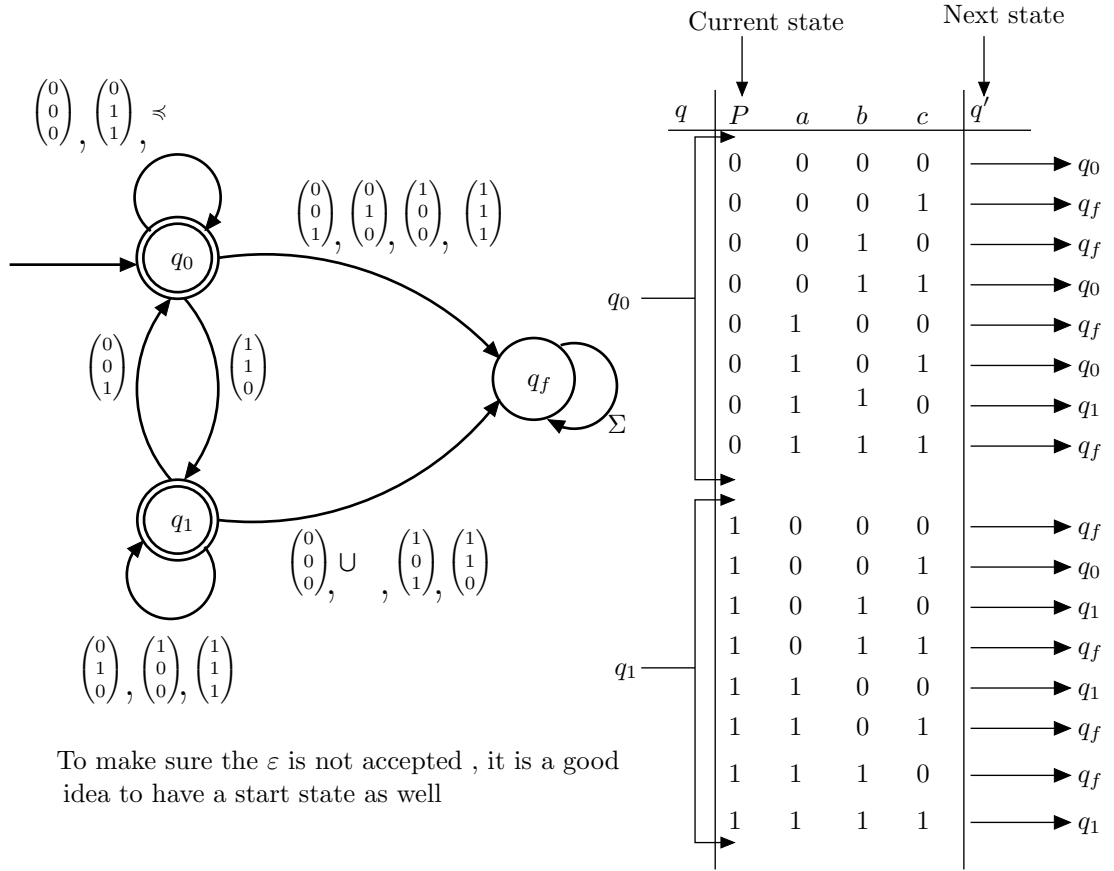
It is easy to see that in j -steps it is possible to derive β from α . i.e

$$\begin{aligned} S \Rightarrow^* x_0 A_1 x_1 \dots x_{j-1} A_j x_j &\Rightarrow x_0 \gamma_1 x_1 \dots x_{j-1} A_j x_j \\ &\Rightarrow \left. \begin{array}{c} \vdots \\ \Rightarrow x_0 \gamma_1 x_1 \dots x_{j-1} A_j x_j \end{array} \right\} j\text{-steps} \end{aligned}$$

Hence β is a sentential form of the grammar □

Corollary 3. (*Converse of yield theorem*) *The yield of any derivation tree is a sentence of $\mathcal{L}(G)$.*

Q1. We need to distinguish between states q_0 and q_1 which generate a carry of 0 or 1. Initially there is only a previous carry of 0. Depending on the carry a triple may be acceptable or not. Once an unacceptable triple arrives the machine goes into a fail state and never recovers. The



To make sure the ε is not accepted, it is a good idea to have a start state as well

Q2. Given $G_1 = \langle V_1, \Sigma, S_1, P_1 \rangle$ and $G_2 = \langle V_2, \Sigma, S_2, P_2 \rangle$ we construct $G_{1 \cup 2} = \langle V_{1 \cup 2}, \Sigma, S_{1 \cup 2} \rangle$ as follows

$$V_{1 \cup 2} = V_1 \cup V_2 \cup \{S_{1 \cup 2}\}$$

$$P_{1 \cup 2} = \{S_{1 \cup 2} \rightarrow S_1 | S_2\} \cup P_1 \cup P_2$$

$G_{12} = \langle V_{12}, \Sigma, S_{12}, P_{12} \rangle$ as follows: Intuitively we generate $\mathcal{L}(G_1)$ and then begin to generate $\mathcal{L}(G_2)$ appended to it. So it would suffice get the desired language. However since this production is not right linear we need to distribute the “appendage” S_2 to all the terminal productions of P_1 . Hence

$$P_{12} = \{A_1 \rightarrow x_1 S_2 \mid A_1 \rightarrow x_1 \in P, x_1 \in \Sigma^*\} \cup P_2$$

with $V_{12} = V_1 \cup V_2$ and $S_{12} \equiv S_1$.

Let $G = \langle V, \Sigma, S, P \rangle$ be right linear grammar. We construct $G_\star = \langle V_\star, \Sigma, S_\star, P_\star \rangle$ as follows .

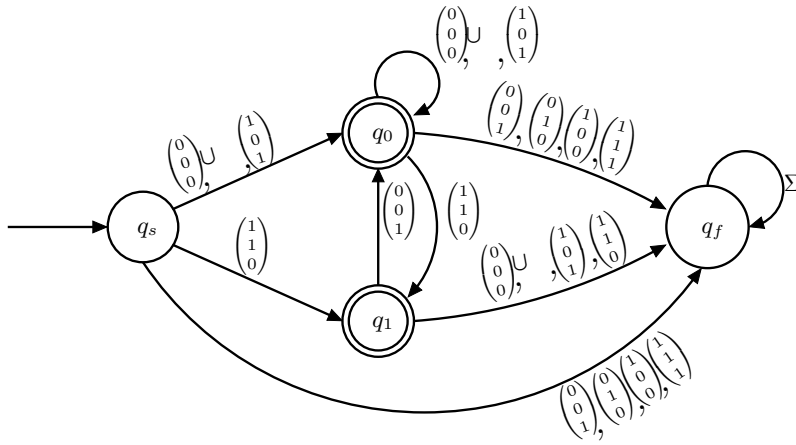
$$P_\star = \{S_\star \rightarrow \varepsilon \mid S\} \cup$$

$$\{A \rightarrow x S_\star \mid A \rightarrow x \in P, x \in \Sigma^*\} \cup$$

$$\{A \rightarrow x B \mid A \rightarrow x B \in P, x \in \Sigma^*\}$$

where $V_\star = V \cup \{S_\star\}$.

Since for right linear grammars all sentential forms generated are of the form $\Rightarrow xA$ where $x \in \Sigma^*$ and $A \in V$, there is always at most one nonterminal at the end of a sentential form. So there is no danger of having forms such as xAS_2yS_2 .



q_1 can't be a final state because the carry it carries cannot be consumed. Hence $\begin{pmatrix} a_n \\ b_n \\ c_n \end{pmatrix}$ should not generate any carry.

Q3 $A = \{0000ww^R \mid w \in \Sigma^*\}$. To show that A is not regular we prove as challenger that for any $m > 0$ chosen by the defender, the challenger has a winning strategy.

DEFENDER : Assume $m > 0$ is chosen.

CHALLENGER : Choose $y \in \Sigma^*$ such that $|y| > m$.

Hence $x = yy^R \in A$ chosen with $|x| > 2m$.

DEFENDER : Since defender decomposes x into 3 parts u, v, w with $v \neq \varepsilon$ and $|uv| \leq m$, any decomposition of the defender such that $x = yy^R = uvw$ necessarily decomposes y into 4 parts $y = uvz$. Hence

$$x = yy^R = uvzz^Rv^Ru^R \text{ and } w = zz^Rv^Ru^R.$$

CHALLENGER : Choose $k = 2$ with

$$\begin{aligned} x_R = x_2 &= uv^2w = uvvw \\ &= uvvzz^Rv^Ru^R \\ &= uvvz(vz)^Ru^R \notin A. \end{aligned}$$

Note : $x = \alpha\beta$
 $\Leftrightarrow x^R = \beta^R\alpha^R$

Hence A is not regular.

Note that A is not regular only if $|\Sigma| > 1$

If $|\Sigma| = 1$ i.e. $\Sigma = \{a\}$ then $A = \{a^{2n} \mid n \geq 0\}$ which is regular.

Othersolutions involve choosing $a \neq b$ and $x = a^m b b a^m$ and proceeding with the argument.

Q4 It is very difficult to give anything except an (inequational) characterization of $r \parallel s$ which we give below starting from structural rules and choosing under commutatively $r \parallel s = s \parallel r$

$$\phi \parallel s = \phi$$

$$\varepsilon \parallel s = s$$

$$a.r' \parallel s = a.(r' \parallel s) + s \parallel (a.r')$$

$$(r_1 + r_2) \parallel s = r_1 \parallel s + r_2 \parallel s$$

$$r^* \parallel s = s + (r_1.r_2) \parallel s$$

finally we have the rule

$$r_1.r_2 \leq \Rightarrow (r_1.r_2) \parallel s \leq s$$

$$\text{and } r_1.r_2 \geq r \Rightarrow (r_1.r_2) \parallel s \geq r \parallel s.$$

where \leq is the partial order on regular expressions induced by the inequation

$$\boxed{r \leq r + s}$$

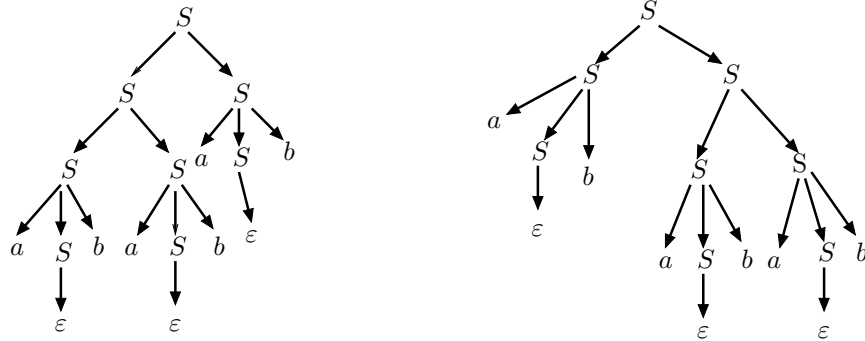
and \geq is \leq^{-1} .

0.1 AMBIGUITY & PARSING

We have stated that the grammar

$$S \longrightarrow SS \mid aSb \mid \varepsilon$$

is ambiguous. For example the sentence $abab$ has two possible derivation trees.



The ambiguity in this case has to do with how we group “chunks” of balanced parenthesis where a “chunk” is a maximal substring of the form $a^k b^k$. Since concatenation is associative there are different ways of grouping the chunks into sequences of chunks — either $(ab ab)ab$ or $ab(ab ab)$. In this particular case we may resolve the ambiguity by choosing a “regular” to denote a chunk and use it to generate tail recursive sequences of chunks as follows.

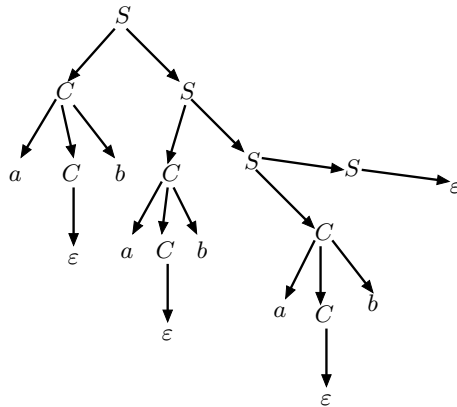
$$S \longrightarrow \varepsilon \mid CS$$

$$C \longrightarrow aSb$$

↑

Within each chunk there may be sequences too !

With this grammar the sequence “ $ababdb$ ” has only one derivation tree as follows.



This new variable with a “determining” effect adds extra control.

The language generated by these grammars is BP . Suppose we need to generate $BP^+ = BP - \{\varepsilon\}$. We could proceed as follows.

$$\begin{aligned} S &\longrightarrow C \mid CS \\ C &\longrightarrow ab \mid aSb. \end{aligned}$$

Exercise

1. Prove that these two grammars generate BP^+ .
2. How do you show that this grammar is unambiguous?

Parsing & Membership

A crucial decision problem in compilers is the question of “whether a given string $x \in \Sigma^*$ belongs to $\mathcal{L}(G)$ ” for a grammar. This is the membership problem which gets solved automatically if there exists an algorithm to determine “whether there exists a derivation tree for x ” or equivalently “whether there exists a derivation $S \Rightarrow^* x$ ” equivalently “whether there exists a sequence of production applications which will derive x from S ”. This is called a “parse”.

There exists an exhaustive method of determining whether there is a parse of $x \in \Sigma^*$.

Definition 20. Given a CFG G , a production $A \longrightarrow \varepsilon$ is called a null production, and any production of the form $A \longrightarrow B$ for $A, B \in V$ is called a unit production². A CEG which has no null productions is called a positive CFG. A positive CFG which has no unit productions is called a branching CFG.

Facts

1. Every production of a positive CFG is length non-decreasing i.e. $A \longrightarrow \alpha \in (V \cup \Sigma)^+$ implies $|\alpha| \geq 1$. Hence for any derivation $S \Rightarrow^* \alpha_i \Rightarrow \alpha_{i+1}$ it is guaranteed that $|\alpha_i| \leq |\alpha_{i+1}|$.
2. In a branching CFG every production except terminal ones is strictly length increasing. Hence in any derivation.

$$S \Rightarrow^* \alpha_i \Rightarrow \alpha_{i+1} \Rightarrow \dots$$

$|\alpha_i| = |\alpha_{i+1}|$ iff α_{i+1} is obtained from α_i by application of a length-preserving terminal production.

$|\alpha_i| < |\alpha_{i+1}|$ iff α_{i+1} is obtained from α_i by application of a length-increasing production.

Theorem 21. If G is a branching CFG then the parsing problem for G is solvable i.e. There exists an algorithm which for any $x \in \Sigma^*$ can decide membership and if $x \in \mathcal{L}(G)$ it produces a parse of x .

Proof.

Claim 16. For each steps $\alpha_i \Rightarrow \alpha_{i+1}$ in a derivation either $|\alpha_i| < |\alpha_{i+1}|$ or the number of terminal symbols in α_i is less than the number of terminal symbols in α_{i+1} .

Claim 17. Because of claim 1, the number of steps in any derivation $S \Rightarrow^* x$ is at most $2|x|$.

Claim 18. There are only a finite number of leftmost derivations of any given length

²a terminal production is one of the form $A \longrightarrow a$.

Proof. In each derivation steps of a leftmost from a given sentential from α_i there are atmost $|P|$ different α_{i+1} , since only one of the productions may be applied to a leftmost non-terminal in α_i to obtain α_{i+1} . Hence any derivation

$$S \Rightarrow \alpha_1 \Rightarrow \dots \alpha_m$$

allows for at most $|P|$ possibilities for α_1 and for each possibilty α_1 , there are at most $|P|$ possibilities of α_2 , which implies there are only $|P|^2$ different leftmost derivation. Proceeding in this fashion for any partial derivation of length k . there are at most $|P|^k$ different possibilities. \square

Exhaustively enumerate all partial derivations of length $\leq 2|x|$. $x \in \mathcal{L}(G)$ iff x is the string generated by one of the derivations and if so the leftmost derivation that generates x also yields a parse of x in the grammar. \square

The previous theorem indicates that the parsing problem for general *CFG* would be solvable provided

- (i) it is possible to eliminate unit productions and
- (ii) it is possible to isolate ε productions and transform the grammar into a branching grammar. In effect we would like to prove the following theorems.

Theorem 22. *A language $A \subseteq \Sigma^*$ is context-free iff there is a positive grammar G such that $\mathcal{L}(G) = A - \{\varepsilon\}$. Moreover any *CFG* G may be transformed into a positive *CFG*, G^+ such that $\mathcal{L}(G^+) = \mathcal{L}(G) - \{\varepsilon\}$.*

Theorem 23. *Any positive context-free grammar may be transformed into an equivalent branching context-free grammar.*

Transformations of *CFGs*

The substitution Rule. Let $G = \langle V, \Sigma, S, P \rangle$ be a *CFG*.

Lemma 14. *If $A \rightarrow \alpha B \beta \in P$ for $A \neq B$ and $B \rightarrow \Gamma_1 | \dots | \Gamma_k$ are all productions of B . Then $\widehat{G} = \langle V, \Sigma, S, \widehat{P} \rangle$ where $\widehat{P} = P - \{A \rightarrow \alpha B \beta\} \cup \{A \rightarrow \alpha \Gamma_i \beta \mid 1 \leq i \leq k\}$ is an equivalent grammar.*

Proof. Consider any derivation in G such that

$$S \Rightarrow_G \star \rho A \sigma \Rightarrow_G \rho \alpha B \beta \sigma \Rightarrow_G \dots \Rightarrow_G \tau B \Theta \Rightarrow_G \tau \Gamma_i \Theta \Rightarrow \dots$$

where $\rho \alpha \Rightarrow \star \tau$ and $\beta \rho \Rightarrow \star \Theta$. A corresponding derivation in \widehat{G} exists with

$$S \Rightarrow_{\widehat{G}} \star \rho A \sigma \Rightarrow \rho \alpha \Gamma_i \beta \sigma \Rightarrow \dots \Rightarrow \tau \Gamma_i \Theta \Rightarrow \dots$$

Since $\rho \alpha \Rightarrow \star \tau$ and $\beta \rho \Rightarrow \star \Theta$. Hence

$$S \Rightarrow_G \star x \Rightarrow S \Rightarrow_G \star x. \quad \text{i.e.} \quad \mathcal{L}(G) \subseteq \mathcal{L}(\widehat{G})$$

Similarly for any derivation in \widehat{G} such that

$$S \Rightarrow_{\widehat{G}} \star \rho A \sigma \Rightarrow_{\widehat{G}} \rho \alpha \Gamma_i \beta \sigma \Rightarrow_{\widehat{G}} \dots$$

there exists a corresponding derivation

$$S \Rightarrow_G \star \rho A \sigma \Rightarrow \rho \alpha B \beta \sigma \Rightarrow \rho \alpha \Gamma_i \beta \rho \Rightarrow \dots$$

Hence $S \Rightarrow_{\widehat{G}} \star x \Rightarrow S \Rightarrow_G \star x$ i.e. $\mathcal{L}(\widehat{G}) \subseteq \mathcal{L}(G)$. \square

Example. $G : S \rightarrow \varepsilon \mid cs, c \rightarrow aSb$
 $\widehat{G} : S \rightarrow \varepsilon \mid aSbS$

As a converse of the substitution rule we could have a factoring rule which allows a rule $A \rightarrow \alpha\beta\Gamma \in P$ to be replaced by $A \rightarrow \alpha \times \Gamma$ and a new rule with $X \notin V$ introduced as $X \rightarrow \beta$.

Factoring allows us to transform the grammar $\widehat{G} : S \rightarrow \varepsilon \mid aSbS$ to the grammar $G : S \rightarrow \varepsilon \mid CS \rightarrow aSb$

Removal of Useless productions

Definition 21. a variable $A \in V$ is useful iff

$$\exists x \in \mathcal{L}(G) : S \Rightarrow^* \alpha A \beta \Rightarrow^* x$$

otherwise a variable is called useless³. A production is useless if it involves a useless variable (either on the LHS or in the RHS).

Example. $S \rightarrow A$, $A \rightarrow aA \mid \varepsilon$, $B \rightarrow bA$. Here B is a useless variable and $B \rightarrow bA$ is a useless production.

Theorem 24. Let $G = \langle V, \Sigma, S, P \rangle$ be a CFG. There exists an equivalent CFG , $\widehat{G} = \langle \widehat{V}, \widehat{\Sigma}, S, \widehat{P} \rangle$ which contains no useless productions.

Theorem 25. There exists an algorithm to construct $V_1 \subseteq V$ such that $V_1 = \{A \in V \mid A \Rightarrow^* x \in \Sigma^*\}$.

³A variable is useless if either it can't be reached from S or it can't produce a terminal string

Algorithm 1

$$V_{1\text{ old}} := \phi; V_1 := \{A \in V \mid \exists x \in \Sigma^* : A \rightarrow x \in P\}$$
$$\boxed{INV : \forall A \in V_1 : A \Rightarrow \star x \text{ for some } x \in \Sigma^*}$$
while $V_{1\text{ old}} \neq V_1$ **do**
$$\left\{ \begin{array}{l} V_{1\text{ old}} := V_1 ; \\ V_1 := V_1 \cup \{A \in V \mid \exists A \rightarrow \alpha \in P : \text{variable}(\alpha) \subseteq V_1\} \\ P_1 := \{A \rightarrow \alpha \in P \mid \text{symbols}(A \rightarrow \alpha) \subseteq V_1 \cup \Sigma\} \end{array} \right\}$$
end while

Claim 19. *There exists an algorithm to construct $\widehat{V} \subseteq V_1$ such that $\widehat{V} = \{A \in V_1 \mid S \Rightarrow \star \dots A \dots\}$.*

Algorithm 2

$$\widehat{V}_{\text{old}} := \phi; \widehat{V}_1 := \{A \in V_1 \mid S \rightarrow \dots A \dots \in P_1\}$$
while $\widehat{V}_{\text{old}} \neq \widehat{V}$ **do**
$$\left\{ \begin{array}{l} \widehat{V}_{\text{old}} := \widehat{V} \\ \text{foreach } B \in V_1 \\ \widehat{V} := \widehat{V} \cup \{B \in V_1 \mid \exists A \in \widehat{V} : \exists A \rightarrow \dots B \dots \in P_1\} \\ \widehat{P} := \{A \rightarrow \alpha \in P_1 \mid \text{symbols}(A \rightarrow \alpha) \subseteq \widehat{V} \cup \Sigma\} \end{array} \right\}$$
end while

It is clear that \widehat{V} consists of only useful variables and \widehat{P} only useful productions from the grammar. It is also possible to extend the algorithms to construct Σ , and $\widehat{\Sigma}$ which gives only the useful terminal symbols

PROOF OF UNAMBIGUITY

To prove that the grammar

$$\boxed{S \rightarrow \varepsilon \mid CS \qquad C \rightarrow aSb}$$

is unambiguous

$$\boxed{\begin{array}{l} C \Rightarrow^* x \Rightarrow \#a(x) = \#b(x) \wedge \\ \forall y \preceq x : \#a(y) \geq \#b(y) \\ \hline S \Rightarrow^* x \Rightarrow \#a(x) = \#b(x) \wedge \\ \forall y \preceq x : \#a(y) \geq \#b(y) \end{array}}$$

By the substitution lemma

$$G : S \longrightarrow \varepsilon | aSbS$$

Proof by contradiction. Assume $\mathcal{L}(G)$ is ambiguous.

$\Rightarrow \exists \in \mathcal{L}(G) : x$ has two distinct leftmost derivations.

Let $x \in \mathcal{L}(G)$ be the smallest length word that is ambiguous $\Rightarrow \boxed{x \neq \varepsilon}$

i.e. $\forall y \in \mathcal{L}(G) : |y| < |x| \Rightarrow y$ has a unique leftmost derivation

$x \neq \varepsilon \Rightarrow S \Rightarrow aSbS \Rightarrow^* aubv = x$ where $|u| < |x|$ and $|v| < |x|$.

But by IH $S \Rightarrow^* u$ and $S \Rightarrow^* v$ both have unique leftmost derivation.

For allm derivation of x , $\varepsilon \neq x \in \mathcal{L}(G)$ there is a unique starting point viz.

$$S \Rightarrow aSbS.$$

Since x has \leq two leftmost derivations and both derivations come from $aSbS$, there must exists u', v' such, $u', v', \in \mathcal{L}G$

$$x = au'bv' = aubv$$

but $u' \neq u$

Claim 20. $u \neq u' \Rightarrow u \prec u'$ or $u' \prec u$

Facts. $u, u', v, v' \in \mathcal{L}(G)$

Proof. $u \neq u' \Rightarrow v \neq v'$. Since

$$x = aubv = au'bv'$$

$$\Rightarrow ubv = u'bv'$$

WLOG assume $u \prec u'$

$$\Rightarrow u' = uy \text{ with } |y| > 0. (y \neq \varepsilon)$$

$$\Rightarrow ubv = uybv'$$

$$\Rightarrow bv = ybv'$$

$$\Rightarrow y(1) = b \quad \wedge \quad y = bz \text{ for some } |z| \geq 0$$

But this is a contradiction.

Since $y \neq \varepsilon$, $y \in \mathcal{L}(G) \Rightarrow y = ay'$

Hence the assumption $u \prec u'$ is false. Similarly we may prove that the assumption $u' \prec u$ is also false. \square

Definition 22. Any production of the $A \longrightarrow \varepsilon$ is a null production and a variable $A \in V$ is called nullable if $A \Rightarrow^* \varepsilon$.

Fact. $A \longrightarrow \varepsilon$ implies A is nullable.

Lemma 15. There exists an algorithm to construct the set of all nullable variables.

Proof. \square

Lemma 16. There exists an algorithm to transform a given CFG, $G = \langle V, \Sigma, S, P \rangle$ into a positive CFG, $G^+ = \langle V, \Sigma, S, P^+ \rangle$ such that $\mathcal{L}(G^+) = \mathcal{L}(G) - \{\varepsilon\}$.

Proof. \square

Claim 21. $\mathcal{L}(G^+) \subseteq \mathcal{L}(G)$

Claim 22. For any $A \in V$, if $A \Rightarrow_{G^+}^* x \neq \varepsilon$ then $A \Rightarrow_{G^+}^* x$

Algorithm 3

$V_{N\ old} := \phi$;
 $V_N := \{A \in V \mid A \longrightarrow \varepsilon \in P\}$;
while $V_{N\ old} \neq V_N$ **do**
 {
 $V_{N\ old} := V_N$;
 foreach $A \longrightarrow B_1 \dots B_k \in P : \{B_1, \dots, B_k\} \subseteq V_{N\ old}$ **do**
 $\{V_N := V_N \cup \{A\}\}$
 }
end while

Algorithm 4

1. Compute V_N the set of all nullable variables.
2. $P_0^+ = P - \{A \longrightarrow \varepsilon \mid A \longrightarrow \varepsilon \in P\}$;
 foreach $A \longrightarrow \alpha \in P_0^+ : \alpha = \alpha B_1 \alpha_1 B_2 \dots \alpha_{n-1} B_n \alpha_n \ \wedge$
 $variables(\alpha_0 \alpha_1 \dots \alpha_n) \cap V_N = \phi \ \wedge$
 $\forall i : 1 \leq i \leq k : B_i \in V_N$ **do**
 {
 foreach sequence $s : \{1, \dots, b\} \longrightarrow \{0, 1\}$:
 $P^+ := P_0^+ \cup \{A \longrightarrow \alpha s \mid s : \{1, \dots, n\} \longrightarrow \{0, 1\}\}$
 where the sequence s determines which variable occurrences from α must be deleted to obtain αs .
 }
}

Proof. By induction on the length of the derivation $A \Rightarrow_G^* x$. □

Theorem 26. A language $A \subseteq \Sigma^*$ is context-free iff there exists a positive context-free grammar $G^+ = \langle V, \Sigma, S, P^+ \rangle$ with $\mathcal{L}(G^+) = A - \{\varepsilon\}$.

Proof. (\Leftarrow) If $\mathcal{L}(G^+) = A - \{\varepsilon\}$ then $G = \langle V, \Sigma, S, P^+ \cup \{S \longrightarrow \varepsilon\}^4 \rangle$
(\Rightarrow) from the previous lemma. □

Unit productions

Definition 23. Any production of the form $A \longrightarrow B$, $A, B \in V$ is called a unit production.

Fact. A positive grammar with no unit productions is a branching grammar.

Theorem 27. Let $G^+ = \langle V, \Sigma, S, P^+ \rangle$ be a positive CFG. Then there exists a branching grammar $\widehat{G} = \langle V, \Sigma, S, \widehat{P} \rangle$ which generates the same language.

Proof. Clearly $G_1^+ = \langle V, \Sigma, S, P_1^+ \rangle$ with

$$P_1^+ = P^+ - \{A \longrightarrow A \mid A \in V\}$$

generates the same language as G^+ .

Consider all unit productions $A \longrightarrow B \in P_1^+$ with $A \neq B$. Now construct a dependency graph to find all $A \neq B$ such that $A \Rightarrow^* B$.

⁴This would lead to trouble with accuring on the RHS of a production.000

$$\text{Let } \widehat{P} = \{A \longrightarrow \alpha \mid |\alpha| \geq 1, \alpha \notin V\} \cup \\ \{A \longrightarrow \Gamma_1 | \Gamma_2 | \dots | \Gamma_k \mid B \longrightarrow \Gamma_1 \dots | \Gamma_k \text{ in } \widehat{P}\}.$$

there are the only rules of B in \widehat{P}

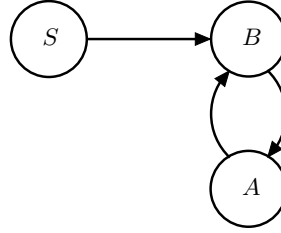
□

Claim 23. $\mathcal{L}(\widehat{G}) = \mathcal{L}(G^+)$.

Proof. Follows from the substitution lemma

□

Example 1. $S \longrightarrow Aa \mid B$
 $B \longrightarrow A \mid bb$
 $A \longrightarrow a \mid bc \mid B$



$$\begin{aligned} S &\Rightarrow^* A \\ S &\Rightarrow^* B \\ A &\Rightarrow^* B \\ B &\Rightarrow^* A \end{aligned}$$

$$\widehat{P} = \{S \longrightarrow Aa, B \longrightarrow bb, A \longrightarrow a|bc\} \cup \\ \{S \longrightarrow a|bc|bb, A \longrightarrow bb, B \longrightarrow a|bc\}.$$

Note that the removal of unit productions has made B useless.

Theorem 28. For every CFG, $G = \langle V, \Sigma, S, P \rangle$ there exists a branching CFG, $\widehat{G} = \langle \widehat{V}, \Sigma, S, \widehat{P} \rangle$ such that $\mathcal{L}(\widehat{G}) = \mathcal{L}(G) - \{\varepsilon\}$ and \widehat{G} has no useless productions, no null productions and no unit productions.

Proof. We note the following properties of our transformations.

1. The removal of ε -productions may create unit productions.
2. The removal of unit - productions may create useless productions.

However, it is also true that

1. The removal of unit productions does not create any null productions.
2. The removal of useless productions does not create any fresh null or unit productions.

Hence the following algorithm proves the theorem.

Algorithm 5

- 1: Remove all ε -productions (to obtain a positive grammar G^+)
 - 2: Remove all unit productions (to obtain a branching grammar \widehat{G}_1)
 - 3: Remove all useless productions (to obtain a branching grammar \widehat{G}_1)
-

□

NORMAL FORMS

Definition 24. A CFG is in Chomsky Normal form (CNF) if all productions are of the form

$$A \longrightarrow BC \quad \text{or} \quad A \longrightarrow a$$

where $A, B, C \in V$ and $a \in \Sigma$. It is in Greibach Normal form (GNF) if each its productions is of the form

$$A \longrightarrow aB_1B_2 \dots B_k$$

for some $k \geq 0$, $A, B_1, B_2, \dots, B_k \in V$.

Fact

1. Every grammar in CNF or GNF is positive and branching. Hence these normal forms do not allow for generation of ε or unit productions.

Theorem 29. Every branching CFG, G may be transformed into an equivalent one in CNF.

Proof. Since G is branching it has

$$\left. \begin{array}{l} \text{(i) no null productions} \\ \text{and (ii) no unit productions} \end{array} \right\} \begin{array}{l} \text{Even if it has we may} \\ \text{assume wlog that they} \\ \text{have been removed.} \end{array}$$

The construction of the equivalent CFG, \widehat{G} in CNF proceeds as follows.

Step 1. Let $G = \langle V, \Sigma, S, P \rangle$ be a branching CFG. Let $G' = \langle V', \Sigma, S, P' \rangle$ be a CFG obtained from G as follows.

(i) $\forall a \in \Sigma$, let X_a be a new nonterminal ($X_a \notin V$)

$V' = V \cup X_\Sigma$ where $X_\Sigma = \{X_a | a \in \Sigma\}$.

$\forall a \in \Sigma : X_a \longrightarrow a \in P'$ □

Claim 24. Every production in P' is of the form

$$\boxed{A \longrightarrow a \quad \text{or} \quad A \longrightarrow B_1B_2 \dots B_k, k > 1}$$

and $\mathcal{L}(G') = \mathcal{L}(G)$.

Step 2. Obtain $G'' = \langle V'', \Sigma, S, P'' \rangle$ from G' as follows :

Proof. Let $P_1'' = \{A \longrightarrow a | A \longrightarrow a \in P'\}$.

We construct P_2'' as follows from $P' - P_1''$.

Every production in $P' - P_1''$ is of the form

$$A \longrightarrow B_1B_2 \dots B_k, K > 1.$$

Introduce new variable C_2, \dots, C_{k-1} and let

\widehat{P}_2 contain the following productions

$$\left. \begin{array}{l} A \longrightarrow B_1C_2 \\ C_2 \longrightarrow B_2C_3 \\ \vdots \\ C_{k-2} \longrightarrow B_{k-2}B_{k-1} \\ C_{k-1} \longrightarrow B_{k-1}B_k \end{array} \right\} \begin{array}{l} \text{Repeat this for every} \\ \text{such production in } P' \\ V'' = V' \cup \{C | C \text{ introduced in this step}\} \\ P'' = P_1'' \cup P_2'' \end{array} \quad \square$$

Claim 25. $G'' = \langle V'', \Sigma, S, P'' \rangle$ is in CNF and $\mathcal{L}(G'') = \mathcal{L}(G')$

Step 3. Obtain \widehat{G} from G'' by removing useless productions.

Claim 26. $\mathcal{L}(\widehat{G}) = \mathcal{L}(G'') = \mathcal{L}(G') = \mathcal{L}(G)$

Greibach Normal form

Definition 25. A production is left-recursive if it is of the form $A \rightarrow A\alpha$.

Lemma 17. (left-recursive removal). Let $G = \langle V, \Sigma, S, P \rangle$ be a CFG and let $A \rightarrow A\alpha_1 | \dots | A\alpha_r$ be all the left recursive productions of A and $A \rightarrow \beta_1 | \dots | \beta_s$ be all the other productions of A . Let $G' = \langle V \cup \{B\}, \Sigma, S, P' \rangle$ be a CFG obtained from G by adding a new variable B and replacing all the left-recursive productions of A by the following productions.

$$A \rightarrow \beta B | \dots | \beta_s B.$$

and adding the following productions for B

$$\begin{aligned} B &\rightarrow \alpha_1 \dots | \alpha_r \\ B &\rightarrow \alpha_1 B | \dots | \alpha_r B. \end{aligned}$$

Then $\mathcal{L}(G') = \mathcal{L}(G)$.

Proof. Consider any leftmost derivation of A in G which terminal in a sentential form not containing A .

$$A \Rightarrow A\alpha_{i_1} \Rightarrow A\alpha_{i_2}\alpha_{i_1} \Rightarrow \dots \Rightarrow A\alpha_{i_m} \dots \alpha_{i_1} \Rightarrow \beta_j \alpha_{i_m} \dots \alpha_{i_1}.$$

where $1 \leq j \leq s$ and $1 \leq i_1, i_2, \dots, i_m \leq r$.

This derivation may be replaced by the following derivation in G' .

$$A \Rightarrow \beta_j B \Rightarrow \beta_j \alpha_m B \Rightarrow \beta_j \alpha_{i_m} \alpha_{i_{m-1}} B \Rightarrow \dots \Rightarrow \beta_j \alpha_{i_m} \dots \alpha_{i_2} B \Rightarrow \beta_j \alpha_{i_m} \dots \alpha_{i_2} \alpha_{i_1}$$

(which is not leftmost but is still a derivation). The reverse transformation may also be made. Hence $\mathcal{L}(G) = \mathcal{L}(G')$ \square

NORMAL FORMS (contd.)

Lemma 18. Any CFG in CFG may be transformed into an equivalent CFG in CNF containing no left-recursive productions.

Proof. Let G be a grammar in CNF. The set of production of a variable A may be divided into 3 classes.

class 1 Terminal productions of the form $A \rightarrow a, A \in \Sigma$.

class 2 Non-left-recursive non terminal productions of the form $A \rightarrow BC$ where $A \neq B, A, B, C \in V$

class 3 left-recursive productions of the form

$$A \rightarrow AD \text{ where } A, D \in V$$

Let $A \rightarrow AD_1 | \dots | AD_p$ be all the left recursive productions of A ,

$A \rightarrow B_1 C_1 | \dots | B_n C_n$ be all the non-left-recursive productions

and $A \rightarrow a_1 | \dots | a_m$ be all the terminal productions

The removal of left-recursion involves adding a new variable $E \notin V$, replacing the left-recursive productions of A by

$$A \rightarrow B_1 C_1 E | \dots | B_n C_n E$$

$$| a_1 E | \dots | a_m E$$

and adding the following productions for E

$$E \rightarrow D_1 | \dots | D_p \tag{1}$$

$$| D_1 E | \dots | D_p E \tag{2}$$

and adding the following productions for E

$$E \longrightarrow D_1 | \dots | D_p \quad (3)$$

$$|D_1 E | \dots | a_m E \quad (4)$$

□

Claim 27. Let $G' = \langle V', \Sigma, S, P' \rangle$ where $V' = V \cup \{E\}$ and $P' = (P - \{A \longrightarrow AD_1 | \dots | AD_p\}) \cup \textcircled{1} \cup \textcircled{2} \cup \textcircled{3} \cup \textcircled{4}$ then $\mathcal{L}(G') = \mathcal{L}(G)$.

Claim 28. G' may be transformed into a grammar \widehat{G} in CNF such that $\mathcal{L}(G') = \mathcal{L}(\widehat{G})$.

Proof. Since G' was derived from a CNF G , it is only necessary to transform the productions of the forms $\textcircled{1}, \textcircled{2}, \textcircled{3}, \textcircled{4}$ into CNF.

$\textcircled{1}$ Replace $A \longrightarrow B_1 C_1 E | \dots | B_n C_n E$ by

$$A \longrightarrow B_1 E_1 | \dots | B_n E_n \quad \text{--- } 1'$$

where E_1, \dots, E_n are new variable and add the productions

$$\left. \begin{array}{l} E_1 \longrightarrow C_1 E \\ E_n \longrightarrow C_n E \end{array} \right\} \quad \text{--- } 1''$$

$\textcircled{2}$ Since G was already in CNF there exist variables $X_a \in V$ for each $a \in \Sigma$ and the productions

$$X_a \longrightarrow a \text{ for each } a \in \Sigma \text{ in } G.$$

Replace all the productions $A \longrightarrow X_{a_1} E | \dots | X_{a_m} E \quad \text{--- } 2'$

$\textcircled{3}$ Since D_1, \dots, D_m are variables belonging to the original CNF G , their productions in G

$$\begin{array}{l} D_1 \longrightarrow \delta_{11} | \dots | \delta_{1k_1} \\ \vdots \\ D_m \longrightarrow \delta_{m1} | \dots | \delta_{mk_m} \end{array}$$

are already in the required form for a CNF. Hence by replacing the productions $E \longrightarrow D_1 | \dots | D_p$ by the set of productions.

$$\left. \begin{array}{l} E_1 \longrightarrow \delta_{11} | \dots | \delta_{1k_1} \\ | \\ \vdots \\ | \delta_{m1} | \dots | \delta_{mk_m} \end{array} \right\} \quad \text{--- } 1''$$

we obtain the productions $\textcircled{3}$ of E to conform to CNF.

$\textcircled{4}$ These productions already conform to the CNF.

Hence $\widehat{G} = \langle \widehat{V}, \Sigma, S, \widehat{P} \rangle$ with

$$\widehat{V} = V' \cup \{E_1, \dots, E_n\}.$$

$\widehat{P} = (P - \{A \longrightarrow AD_1 | \dots | AD_p\}) \cup 1' \cup 1'' \cup 2' \cup 3' \cup \textcircled{4}$ is a grammar in CNF which is equivalent to G' . □

Definition 26. A CFG G has the increasing index property (IIP) if there exists a total ordering $<$ on the variables such that for every production of the form $A \longrightarrow B\alpha$ for $A, B \in V$, $A < B$. A production satisfies the IIP if it is either of the form $\alpha \longrightarrow a\alpha$ for $a \in \Sigma$ or it is of the form $A \longrightarrow B\alpha$ with $B \in V$ and $A < B$.

Lemma 19. Let G be a CFG in CFG with no left recursive productions. There exists an equivalent grammar \widehat{G} in CNF and an ordering on the variables of \widehat{G} such that \widehat{G} has the IIP.

Proof. Since G is in CNF we may assume without loss of generality that for every $a \in \Sigma$ (which occurs in a sentence of $\mathcal{L}(g)$) there exists an unique associated variable X_a and a unique production $X_a \rightarrow a$. Even if there is a variable X_{ab} (with $X_{ab} \rightarrow a|b$) which has more then 1 terminal production , it may be split into as many different variables as the number of terminals. Correspondingly all occurences of X_{ab} in any production may be spilt into two productions (one for each terminal). \square

Claim 29. V may be partitioned into two disjoint sets

$$V = N \cup X_\Sigma$$

where $X_\Sigma = X_a | a \in \Sigma$ and $N = V - X_\Sigma$.

The ordering: Define a total ordering that satisfies the condition.

$$\forall A \in N. \forall X_a \in X_\Sigma : A < X_a.$$

Transforming G : If G already satisfies the IIP there is nothing to prove. Otherwise assume for convenience that $V = \{A_1, \dots, A_m\}$ are all the variables of V with $\text{index}(A_i) = i$ and ordered by the usual $<$ on integers.

Let $k, 1 \leq k \leq m$ be the smallest index which violates the IIP, i.e there exists a production.

$A_k \rightarrow A_j A_l \quad , \quad k > j$	--- ①
and $\forall i : 1 \leq i < k$: all the productions of A_i have the IIP. In particular, all the productions of A_j viz.	
PROCESS I $A_j \rightarrow A_l A_{l'_1} \dots A_{l'_m} A_{l'_m}$	
have IIP i.e. $ a_1 \dots a_q $ $j < l_1, l_2, \dots, l_m$	
Repalce $A_r \rightarrow A_j A_l$ by the productions	
$A_k \rightarrow A_{l_1} A_{l'_1} A_l \dots A_{l'_m} A_{l'_m} A_l$ $ a_1 A_l \dots a_q A_l$	--- ②

Repeat PROCESS I for all A_k -productions that violate IIP.

case $k > \min(l_1, \dots, l_m)$. However since $j < \min(l_1, \dots, l_m)$ we have

$j < \min(l_1, \dots, l_m) < k \Rightarrow k - j > k - \min(l_1, \dots, l_m)$. which implies PROCESS I has reduced the value of.

Repeat process I as many times as necessary till either all productions of A_k have the IIP or there are some left-recursive productions of A_k .

Case : All productions of A_k have IIP. Then

Claim 30. The smallest index that violates IIP in the grammar is $> k$

Case. There are left-recursive productions of A_k but all non-left-recursive productions satisfy IIP.

Remove left-recursion in each case by introducing a new variable B and extend the ordering $<$ so that $B <$ any existing variable of V .

Claim 31. *The smallest index of the new grammar that may violate IIP is $> k$.*

However the grammar may not be in CNF. The following case takes this into account

Case $k < l_1, \dots, l_m, \dots$. Then the smallest index that violates IIP is $> k$. However the grammar is no longer in CNF. Let the new productions of A_k be

$$A_k \longrightarrow A_{\alpha_1} | \dots | A_{l_n} \alpha_n \quad \text{where} \quad |\alpha_1| = |\alpha_2| = \dots = |\alpha_n| = 2 \\ |a_1 A_{l_1}| \dots |a_p A_{l_p} \quad \text{and} \quad \forall i : 1 \leq i \leq n : \alpha_i \in V^2.$$

Introduce n new variables B_1, \dots, B_n and replace the above productions by

$$\boxed{A_k \longrightarrow A_{l_1} B_1 | \dots | A_{l_n} B_n \\ |X_{a_1} A_{l_1}| \dots |X_{a_p} A_{l_p}}$$

and add the new productions

$$\boxed{B_1 \longrightarrow \alpha_1 \\ \vdots \quad \text{where} \quad \forall i : 1 \leq i \leq n : \alpha_i \in V^2. \\ B_n \longrightarrow \alpha_n}$$

Extend the total ordering $<$ to include the new variables B_1, \dots, B_n such that

$$\forall i, j : B_i < B_j$$

Claim 32. *The grammar obtained above is in CNF with the smallest index violating the IIP $> k$.*

From the above two cases it follows that the difference $(m - k)$ reduces by the above processes. Hence the above processes need to be repeated only at most $(m - k)$ times to obtain an equivalent grammar in CNF with the IIP

Theorem 30. (GNF) *Every branching CFG may be transformed into an equivalent one in GNF.*

Proof. WLOG we may assume that G is a CNF with the IIP. Let $V = \{A_1, \dots, A_m\}$ be the variables in increasing order of index in the total ordering.

Claim 33. *A_m has only terminal productions.*

$$\boxed{\text{PROCESS 2} \\ \text{Replace every production of the form} \\ B \longrightarrow A_m C \\ \text{with } B \longrightarrow a_1 C | \dots | a_n C \\ \text{where } A_m \longrightarrow a_1 | \dots | a_n \text{ are the only productions of } A_m}$$

Claim 34. *the highest indexed variable occurring as first symbol on the right hand side of any production is A_{m-1} .*

Claim 35. *Since there is no left recursion all the productions of A_{m-1} are of the form*

$$A_{m-1} \longrightarrow a_1 \alpha_1 | \dots | a_p \alpha_p.$$

where $\alpha_1, \dots, \alpha_p \in V$.

Proof. The replacement of A_m by terminal symbol all prod. implies all productions are either of the form

$$\begin{aligned} & A \longrightarrow a\beta \quad \text{where} \quad \beta \in V^*, a \in \Sigma \\ \text{or} \quad & A \longrightarrow BC \quad \text{where} \quad A < B < A_m. \end{aligned} \quad \square$$

PROCESS 3

By repeating PROCESS 2 for $m - 1$ we get that the highest indexed variable occurring as the first symbol on the RHS of every production is $< m - 1$.

This PROCESS 3 may be repeated (at most) $m - 1$ times to obtain a grammar in which every production is of the form.

$$A \longrightarrow a\alpha \quad \text{with} \quad \alpha \in V^*$$

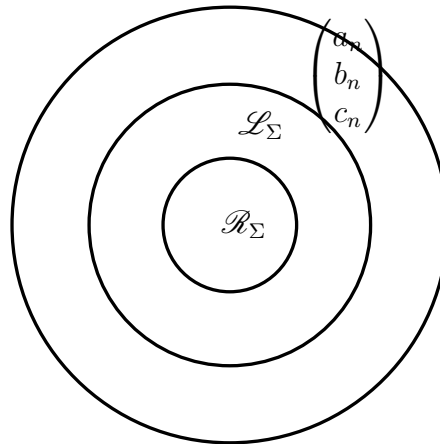
which is in *GNF*. □

1 Pumping lemmas for linear & CFLs

1.1 Pumping Lemma for Linear Languages

A linear language is one which may be generated by a linear grammar i.e. a grammar which may have both right-linear & left linear productions

Fact. Every linear grammar is also context-free with the proof of a pumping lemma for linear languages. We will also show the strict containment of the linear languages within the class of *CFLs*



Theorem 31. Let $A \subseteq \Sigma^*$ be an infinite linear language. Then there exists $m > 0$ such that any $x \in A, |x| \geq m$ may be decomposed into $z = tuvwx$ with

$$\begin{aligned} |tuwx| &\leq m \\ |uw| &\geq 1 \end{aligned}$$

such that for all $i \geq 0, tu_i v w_i x \in A$.

In other words

$$\begin{aligned} & A \subseteq \Sigma^* \text{ is linear} \\ \Rightarrow & \exists m > 0 : \forall z \in A : |z| \geq m : \\ & \exists t, u, v, w, x : z = tuvwx \wedge \overbrace{|tuwx| \leq m} \wedge |uw| \geq 1 \\ & \wedge \forall k \geq 0 : z_k = tu^k v w^k x \in A \end{aligned}$$

Proof. Since the language is linear there exists a linear grammar generating it. As in the case of a *CFG* we may use the same techniques to □

- | | | |
|--|---|---|
| (i) remove unit productions | } | transfer G into a positive branching grammar. |
| (ii) remove ε -productions | | |
| (iii) transfer G to generate $A - \{\varepsilon\}$ | | |

The proof then proceeds as follows:

Consider a derivation tree as in the case of the lemma for *CFLs*. Since the derivation tree may be arbitrarily deep consider only the first and second occurrences of the first variable A that repeats itself in the derivation tree.

Since $|V|$ is bounded it follows that the other variables which do not repeat can only produce only bounded-length terminal strings t and x . Similarly between the two occurrences of A there are only bounded length strings u and w may be produced.

Choose $m = |t| + |u| + |w| + |x|$.

Finally u and w may be pumped as many times as we please to obtain derivations of

$$tu^kvw^kx \in A - \{\varepsilon\}$$

2 The Pumping Lemma for *CFLs*

Theorem 32. *Let $A \subseteq \Sigma^*$ be an infinite CF language. There exists $m > 0$ such that for any $z \in A$ with $|z| \geq m$, z may be decomposed into $z = tuvwx$ with $|uvw| \leq m$, $|uw| \geq 1$ such that for any $k \geq 0$, $tu^kvw^kx \in A$.*

In other words

$ \begin{aligned} & A \subseteq \Sigma^* \text{ is an infinite CFL} \\ \Rightarrow & \exists m > 0 : \forall z \in A : z \geq m : \\ & \quad \exists t, w, x, v : z = tuvwx \wedge uvw \leq m \wedge uw \geq 1 \\ & \quad \wedge \forall k \geq 0 : z_k = tu^kvw^kx \in A \end{aligned} $
--

Proof. Consider $A - \{\varepsilon\}$. Let be a *CFG* with

- | | | |
|--|---|---|
| (i) $\mathcal{L}(G) = A - \{\varepsilon\}$. | } | Let G be a branching <i>CFG</i> with $\mathcal{L}(G) = A - \{\varepsilon\}$. |
| (ii) G has no unit productions | | |
| (iii) G has no ε -productions | | |

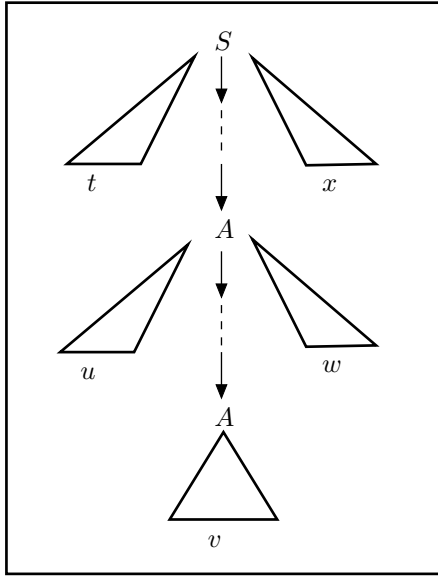
\Rightarrow for every production $A \rightarrow \alpha \in P$

$$\alpha \in \Sigma^+ \vee |A| < |\alpha|.$$

Let $n = \max\{|\alpha| \mid \exists A \in V : A \rightarrow \alpha \in P\}$.

\Rightarrow Any $z \in A - \{\varepsilon\}$ has a derivation of length $\geq \frac{|z|}{n}$. Since $A - \{\varepsilon\}$ is infinite, there exists arbitrarily long derivations and hence there exists derivation trees of arbitrary height. □

We have the following derivation



Consider such a derivation tree of sufficient depth. Since $|V| < \infty$, there must be at least one variable that repeats in the derivation tree.

Let A be the variable that repeats along the path such that no other variable repeats itself between the occurrences of A .

$$S \Rightarrow^* tAx \Rightarrow^* tuAwx \Rightarrow^* tuvwx.$$

where $t, u, v, w, x \in \Sigma^*$. Clearly $A \Rightarrow^* uAw$, is a sub-derivation of the original derivation. We may “pump” this derivation as many times as we want to obtain derivations of $tu^kvw^kx \in A - \{\epsilon\}$. Since G has no unit productions and is positive it follows that $|uw| \geq 1$ and we may choose $m = |uvw| > 1$.

3 Pumping lemma : Contrapositives

$\forall m > 0 : \exists z \in A : z \geq m \wedge \forall u, v, w : z = uvw \wedge uv \leq m \wedge v \geq 1$ $\Rightarrow \exists k \geq 0 : z_k = uv^k w \notin A$
$\Rightarrow A \text{ is not regular}$
$\forall m > 0 : \exists z \in A : z \geq m \wedge \forall u, v, w, x : z = tuvwx \wedge uvw \leq m \wedge uw \geq 1$ $\Rightarrow \exists k \geq 0 : z_k = tu^k vw^k x \notin A$
$\Rightarrow A \text{ is not a CFL}$
$\forall m > 0 : \exists z \in A : z \geq m \wedge \forall t, u, v, w : z = tuvwx \wedge tuvw \leq m \wedge uw \geq 1$ $\Rightarrow \exists k \geq 0 : z_k = tu^k vw^k x \notin A$
$\Rightarrow A \text{ is not linear}$

Examples

1. $a^n b^n c^n \mid n \geq 0$ is not CF. For any $m > 0$ chosen by D, C chooses $a^m b^m c^m$ and does a case analysis.

BEST STRATEGY FOR D TO WIN

Pick a decomposition of $a^m b^m c^m$ such that the string to be pumped viz uw satisfies

$$\#a(uw) = \#b(uw) = \#c(uw)$$

But that is impossible since that implies $uvw = a^i b^m c^i$ and $|uvw| > m$ for $i > 0$ and $|v| = m$ if $i = 0$ we may use the b 's in v to ensure that

$$z_k \notin \{a^n b^n c^n \mid n \geq 0\}.$$

2. $\{ww \mid w \in \{a, b\}^*\}$ is not CF . For any $m > 0$, C chooses $a^m b^m a^m b^m$ and do case analysis.

BEST STRATEGY FOR D TO WIN

Pick a decomposition of $xx = a^m b^m a^m b^m$ such that uvw contains the same length of string from the left x as from the right.

Suppose defender chooses $i > 0$ with $v = \varepsilon$ and $u = b^i$ and $w = a^i$ such that $2i \leq m$. Then it is possible to find $k > 1$ with

$$z_k = a^m b^{mm+i(k-1)} a^{mm+i(k-1)} b^m \notin \{ww \mid w \in \{a, b\}^+\}$$

3. $\{a^n \mid n \geq 0\}$ is not CF Use essentially some argument as for regular case since there is only one terminal symbol.
4. $\{a^n b^m \mid n = m^2\}$ is not CF . C chooses $a^{m^2} b^m$.
5. $\{w \in \{a, b\}^* \mid \#a(w) = b(w)\}$ is not linear⁵ but is CF since $S \rightarrow SS \mid aSb \mid bSa \mid \varepsilon$ generate it.

BEST STRATEGY FOR D TO WIN

Given $z = a^m b^{2m} a^m$, DEFENDER is forced to choose $tuvw = a^i$ for some $i \leq m, i > 0$. Clearly then challenger can choose $k = 0$ and ensure that $\#a(z_0) \neq \#b(z_0)$ and $|uw| \leq 1$. So defender loses

Using Brackets to remove ambiguity in $CFGs$

Let $G = \langle V, \Gamma, S, P \rangle$ be a CFG in CNF . Assume P consists of n rules of the form

$$A \rightarrow BC, \quad 1 \leq i \leq n$$

in addition to terminal productions of the form $A \rightarrow a, a \in \Gamma$.

Define $G_s = \langle V, \Gamma \cup \Pi, S, P_s \rangle$ where

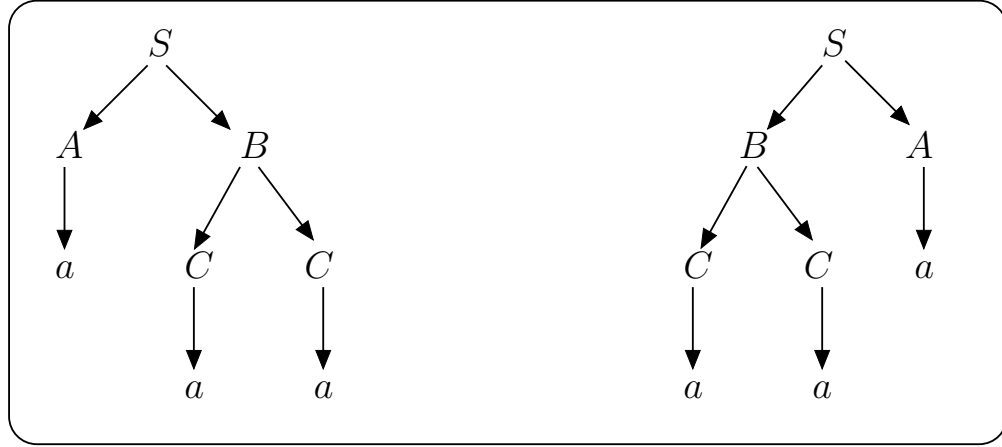
$$P_s = \{A \rightarrow [{}_i B]_i C \mid A \rightarrow BC \text{ is the } i\text{-th rule in } P\} \cup \{A \rightarrow a \mid A \rightarrow a \in P\}.$$

Example. Let G consist of the productions.

$$S \rightarrow AB \mid BA, \quad B \rightarrow CC, \quad A \rightarrow a, \quad C \rightarrow a.$$

The string $aaa \in \mathcal{L}(G)$ has two derivation trees

⁵ C chooses $a^m b^{2m} a^m$



The Chansky-Schutzemberger Theorem

Let $\Sigma = \Gamma \cup \Pi$ where $\Pi = \{ [i,]_i \mid 1 \leq i \leq n \}$ is a collection of n pairs of parentheses of different kinds. The language \prec is generated by the following grammar

$$G_{Pi} \left\{ \begin{array}{l} S \longrightarrow a, \text{ for each } a \in \Gamma \\ S \longrightarrow [i S]_i, \text{ for each } i = 1, 2, \dots, n \\ S \longrightarrow SS \mid \varepsilon \end{array} \right\} \text{ GENERALIZATION OF } BP = PAR_{\{a,b\}}(\phi)$$

Lemma 20. \prec is a CFL satisfying the following properties

1. $\Gamma^* \subseteq \prec$.
2. $\forall x, y \in \prec: xy \in \prec$ i.e. \prec is closed under catenation.
3. $\forall x \in pr: \forall i: 1 \leq i \leq n: [i x]_i \in \prec$.
4. $\forall x \prec:$

$$x \notin \Gamma^* \Rightarrow \exists u, v, w: \exists i: 1 \leq i \leq n: \\ u \in \Gamma^* \wedge u, w \in \prec \wedge \\ x = u [i v]_i w.$$

Proof. 1,2,3 are obvious and so we prove only 4.

4.

Claim 36. $x \notin \Gamma^* \Rightarrow |x| > 1$ since there is at least one pair of parenthesis in x .

\Rightarrow Any derivation of x begins with an application of one of the rules

$$\begin{array}{l} S \longrightarrow [i S]_i, \quad 1 \leq i \leq n \\ S \longrightarrow SS. \end{array}$$

We now proceed by induction assuming the result holds for all strings of length $< |x|$. i.e.

$$\boxed{\begin{array}{l} IH: \forall y \in \prec: |y| < |x| \Rightarrow \\ [y \notin \Gamma \Rightarrow \exists r, s, t: \exists j: 1 \leq j \leq n: \\ r \in \Gamma^* \wedge s, t \in \prec \wedge \\ y = r [j S]_j t \end{array}}$$

Induction steps. Do an analysis of the first step of $S \Rightarrow^* x$.

Case. $S \Rightarrow [{}_i S]_i \Rightarrow^* [{}_i v]_i = x$ where $S \Rightarrow^* v$. The result then follows with $u = \varepsilon = w$.

Case. $S \Rightarrow SS \Rightarrow^* yz = x$ where $S \Rightarrow^* y$ and $S \Rightarrow^* z$. We may safely assume $Y \neq \varepsilon \neq z$, otherwise it is possible to derive x without the first step $S \Rightarrow SS$. Hence $0 < |y| < |x|$ and $0 < |z| < |x|$.

Subcase. $y \in \Gamma^*$. By the IH $z = r[{}_i s]_i t$ and $x \in yr[{}_i s]_i t$ with $yr \in \Gamma^*$, $s, t \in \prec$

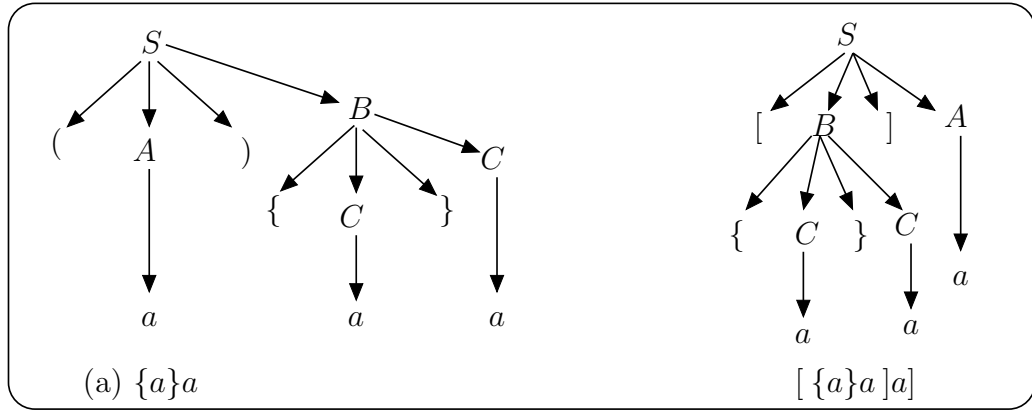
Subcase. $y \notin \Gamma^*$. By the IH $y = r[{}_i s]_i t$ and $x \in r[{}_i s]_i tz$ with $r \in \Gamma^*$, $s, tz \in \prec$

□

The modified grammar G_s has $\pi = \{ (,), [,], \}$. and the productions.

$$S \longrightarrow (A)B \mid [B]A, \quad B \longrightarrow \{C\}C, \quad A \longrightarrow a, \quad C \longrightarrow a.$$

and the corresponding derivation trees generate different strings.



Moral : Bracketing may be used as a means to specify the derivation tree which generates a string

Theorem 33. Let $h : \Gamma \cup \Pi \longrightarrow \Gamma \cup \{\varepsilon\}$ be the homomorphism which maps every symbol of Γ to itself and erases every bracket i.e. $h([{}_i]) = \varepsilon = h(\}_i)$. Then

$$h(\mathcal{L}(G_s)) = \mathcal{L}(G)$$

Proof. Show by induction on the height of derivation trees that both languages have a 1-1 correspondence on their derivation trees □

Lemma 21. $\mathcal{L}(G_s) \subseteq \prec$.

Proof. We prove the stonger result $\mathcal{L}(G_s) \subseteq \{x \mid \exists A \in V : A \Rightarrow^* x\} \subseteq \prec$. By induction on the length of derivations $A \Rightarrow^* x$ for any $A \in V$ and $x \in (\Gamma \cup \Pi)^*$.

Basis. Clearly a terminal production has been applied.

Ind Steps. Otherwise $A \Rightarrow [{}_i B]_i C \Rightarrow^* [{}_i v]_i w = x$ and it follows that there exists smaller derivations $B \Rightarrow^* v$ and $C \Rightarrow^* w$ and it is also clear that x is of the form $u[{}_i v]_i w$ with $u = \varepsilon$, $v, w \in \prec$. By the

IH there exists derivations $S \Rightarrow^* v$ and $S \Rightarrow^* w$ in \prec . Hence we have $S \Rightarrow SS \Rightarrow [{}_i S]_i S \Rightarrow^* [{}_i v]_i w$ in \prec □

Definition 27. Let $G_R = \langle V, \Gamma \cup P_i, S, P_R \rangle$ be the grammar with

$$P_R = \{A \longrightarrow a \mid A \longrightarrow a \in P_s\} \cup \{A \longrightarrow [{}_i B] \mid \exists A \longrightarrow [{}_i B]_i C \in P_s\} \cup \\ A \longrightarrow a]_i C \mid A \longrightarrow a \in P_s, 1 \leq i \leq n \wedge \exists X \longrightarrow [{}_i B]_i C \in P_s$$

Lemma 22. $\mathcal{L}(G_r)$ is regular

Proof. Since the productions are all obviously right linear, $\mathcal{L}(G')$ is regular. □

NOTE : Since G_R is derived from G_s which in term is derived from G which is in *CNF*, it is clear that $\forall A \in V : A \not\Rightarrow^* \varepsilon$.

Lemma 23. We prove that for any $A \in V$, $A \xRightarrow{G_s}^* x \in (\Gamma \cup \Pi)^*$ then $A \xRightarrow{G'}^* x$ by induction on the length l of derivations in G_s .

Basis $l = 1$. Clearly a terminal production.

Ind Step. Let $A \xRightarrow{G_s} [{}_i B]_i \Rightarrow^* x = [{}_i v]_i w$ Clearly $B \xRightarrow{G_R}^* v$ and $C \xRightarrow{G_R}^* w$ by the induction hypothesis. The following derivation in G_R then proves the required result. Clearly since $v \neq \varepsilon$, we may assume $v = ua$ for some $a \in \Gamma$.

Further there must be a productions in G_R of the form there and $\forall i : 1 \leq i \leq n : X_a \longrightarrow a]_i C$ for each $A \longrightarrow [{}_i B]_i C \in P_s$. Hence $B \xRightarrow{G_R} v = ua$ would look like

$$B \xRightarrow{G_R} uX_a \Rightarrow ua = v$$

We may instead use

$$B \xRightarrow{G_R}^* uX_a \xRightarrow{G_R} ua]_i C$$

to obtain the derivation

$$A \xRightarrow{G_R} [{}_i B]_i \xRightarrow{G_R}^* [{}_i uX_a]_i \xRightarrow{G_R} [{}_i ua]_i C \xRightarrow{G_R}^* [{}_i v]_i w$$

Corollary 4. $\mathcal{L}(G_s) \subseteq \prec \cap \mathcal{L}(G_R)$

Lemma 24. $\prec \cap \mathcal{L}(G_R) \subseteq \mathcal{L}(G_s)$

Proof. Consider any derivation $S \xRightarrow{G'}^* x \in \prec$. We need to prove that $S \xRightarrow{G_s}^* x$. We proceed by complete induction on the number of occurrences of symbols from Π in x . (denoted $\#\Pi(x)$).

Basis $\#\Pi(x) = 0$. Clearly a terminal production which also exists in G_s .

Ind steps. Assume the result is true for all strings $y \in \prec \cup \mathcal{L}(G_R)$ with $\#\Pi(y) < \#\Pi(x)$ and let $\#\Pi(y) \geq 0$. Then clearly

$$S \xRightarrow{G_R} [{}_i B] \quad \text{for some } b \in V \text{ and } 1 \leq i \leq n.$$

which implies $x = [{}_i v]_i w$ by the properties of \prec .

Hence

$$S \xRightarrow{G_R} [{}_i B]_i \xRightarrow{G_R}^* [{}_i uX_a]_i \xRightarrow{G_R} [{}_i ua]_i C \xRightarrow{G_R}^* [{}_i v]_i w$$

and by the induction hypothesis we have

$$S \xRightarrow{G_s} [{}_i B]_i C \xRightarrow{G_s}^* [{}_i v]_i w = x$$

□

Corollary 5. $\mathcal{L}(G_s) = \prec \cap \mathcal{L}(G_R)$

Corollary 6. Let $G = \langle V, \Gamma, S, P \rangle$ be a grammar in CNF. Then for a suitably chosen $\Pi \cap \Gamma = \phi$ of pairs of bracketing symbols, there exists a regular language R such that $\mathcal{L}(G) = h(R \cap \mathcal{L}(G_s))$.

Theorem 34. CHOMSKY – SCHUTZEENBERGER. A language $L \subseteq \Gamma^*$ is context-free **if and only if** there is a regular language R and a collection Π of matching bracket symbol pairs such that

$$L = h(R \cap \prec)$$

Proof. (\Rightarrow) For any G in CNF with $\mathcal{L}(G) = L - \varepsilon$, the previous corollary shows the result. If $\varepsilon \in L$, then

$$L = h((R \cup \{\varepsilon\}) \cap \prec).$$

(\Rightarrow) Since \prec is context-free it follows that L is context-free, provided we can show that (i) the intersection of regular and a CFL is a CFL and (ii) homomorphisms preserve CF-ness \square

Theorem 35. The family \mathcal{CF}_Σ is closed union, concatenation and \star -closure

Proof. Let $G_1 = \langle V_1, \Sigma, S_1, P_1 \rangle$ and $G_2 = \langle V_2, \Sigma, S_2, P_2 \rangle$ be CFGs generating $A_1, A_2 \in \mathcal{CF}_\Sigma$ respectively. Assume $V_1 \cap V_2 = \phi$

Union. Let $G_\cup = \langle V_\cup, \Sigma, S_\cup, P_\cup \rangle$ be defined as follows

$$\begin{aligned} V_\cup &= V_1 \cup V_2 \cup \{S_\cup\} \\ P_\cup &= P_1 \cup P_2 \cup \{S_\cup \rightarrow S_1 S_2\}. \end{aligned} \quad \square$$

Claim 37. $\mathcal{L}(G_\cup) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2) = A_1 \cup A_2$

Concatenation. Let $G = \langle V_\bullet, \Sigma, S_\bullet, P_\bullet \rangle$ be the grammar

$$V_\bullet = V_1 \cup V_2 \cup \{S_\bullet\} \quad P_\bullet = P_1 \cup P_2 \cup \{S_\bullet \rightarrow S_1 S_2\}.$$

Claim 38. $\mathcal{L}(G_\bullet) = \mathcal{L}(G_1) \bullet \mathcal{L}(G_2) = A_1 \bullet A_2$

\star -closure. Let $G_\star = \langle V_\star, \Sigma, S_\star, P_\star \rangle$ where

$$V_\star = V_1 \cup \{S_\star\} \quad , \quad P_\star = P_1 \cup \{S_\star \rightarrow \varepsilon | S_1 S_\star\}$$

Claim 39. $\mathcal{L}(G_\star) = \mathcal{L}(G_1)^\star = A_1^\star$

PUSHDOWN AUTOMATA

Definition 28. A nondeterministic push-down automaton (NPDA)

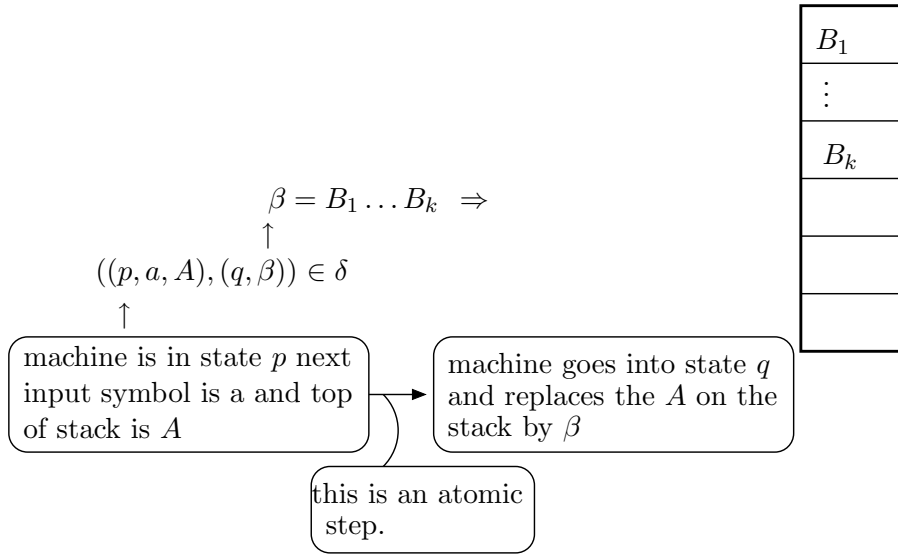
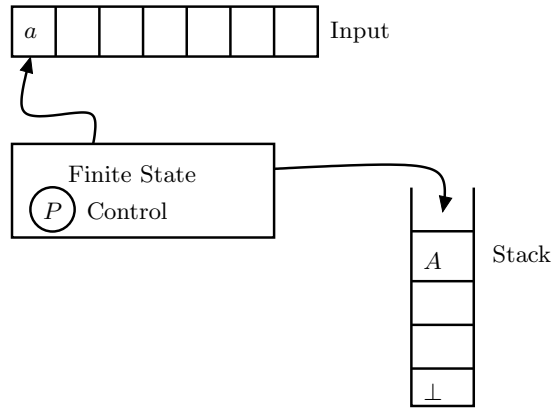
$$\begin{aligned} N &= \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle \\ Q &\text{ is a finite set of states.} \\ \Sigma &\text{ is a finite input alphabet.} \\ \Gamma &\text{ is a finite stack alphabet.} \\ \delta &\subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma) \times (Q \times \Gamma^\star) \text{ is a finite transition relation.} \\ &= Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \xrightarrow{f} \mathbb{2}^{Q \times \Gamma^\star} \end{aligned}$$

$q_0 \in Q$ is the start state.
 $\perp \in \Gamma$ is the initial stack symbol. ($\perp \notin \Sigma$).
 $F \subseteq Q$ is the set of final states.

Note

$$N = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$$

↑

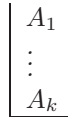


needed only to define the initial configuration

$$Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \xrightarrow{f} 2^{Q \times \Gamma^*}$$

no move is possible if the stack is empty

Configurations. A configuration of the machine is a 3-tuple $\langle q, x, \alpha \rangle$ where $q \in Q$, x is the rest of the input to be consumed and α is the contents of the stack ($\alpha = A_1 \dots A_k \Rightarrow$



$$\begin{aligned} (p, ay, A\beta) &\xrightarrow{N^*} (q, y, \alpha\beta) \text{ if } ((p, a, A), (q, \alpha)) \in \delta \\ (p, ay, A\beta) &\xrightarrow{N^*} (q, y, \alpha\beta) \text{ if } ((p, \varepsilon, A), (q, \alpha)) \in \delta \end{aligned}$$

Acceptance. There are two notions of acceptance. Actually there are two kinds of NPDAs

1. Acceptance by empty stack. (E-acceptance)

$$\mathcal{L}(N_E) = \{x \in \Sigma^* \mid (q_0, x, \perp) \xrightarrow[N_E]^* (q, \varepsilon, \varepsilon)\} \longrightarrow \boxed{\text{Then } F \text{ is irrelevant and may be taken to be } \phi}$$

2. Acceptance by final state (F-acceptance)

$$\mathcal{L}(N) = \{x \in \Sigma^* \mid \exists \gamma \in \Gamma^* : (q_0, x, \perp) \xrightarrow[N_F]^* (q_f, \varepsilon, \gamma), q_f \in F\}$$

Note. But in both cases the input should have been completely consumed.

Example 1. Balanced parantheses. Acceptance by empty attack

$$\left. \begin{array}{l} Q = \{q\} \\ \Sigma = \{[,]\} \\ \Gamma = \{\perp, []\} \end{array} \right\} \begin{array}{l} ((q, [, \perp), (q, [\perp])) \\ ((q, [, []), (q, [])) \\ ((q,], []), (q, \varepsilon)) \\ ((q, \varepsilon, \perp), (q, \varepsilon)) \end{array} \text{ deterministic}$$

Example 2. $A = \{w \in \{a, b\}^* \mid \#a(w) = \#b(w)\}$

$$\left. \begin{array}{l} Q = \{q\} \\ \Sigma = \{a, b\} \\ \Gamma = \{A^6, B^7, \perp\} \end{array} \right\} \begin{array}{l} ((q, \varepsilon, \perp), (q, \varepsilon)) \\ ((q, a, \perp), (q, A)) \\ ((q, \varepsilon, \perp), (q, \varepsilon)) \\ ((q, b, \perp), (q, B)) \\ ((q, a, A), (q, AA)) \\ ((q, a, B), (q, \varepsilon)) \\ ((q, b, A), (q, \varepsilon)) \\ ((q, b, \perp), (q, BB)) \end{array} \begin{array}{l} \text{At any stage the stack contains} \\ \text{(i) only } A\text{'s or} \\ \text{(ii) only } B\text{' or} \\ \text{(iii) neither } A\text{'s nor } B\text{'s} \end{array}$$

The stack would not contain a mix of A's & B's

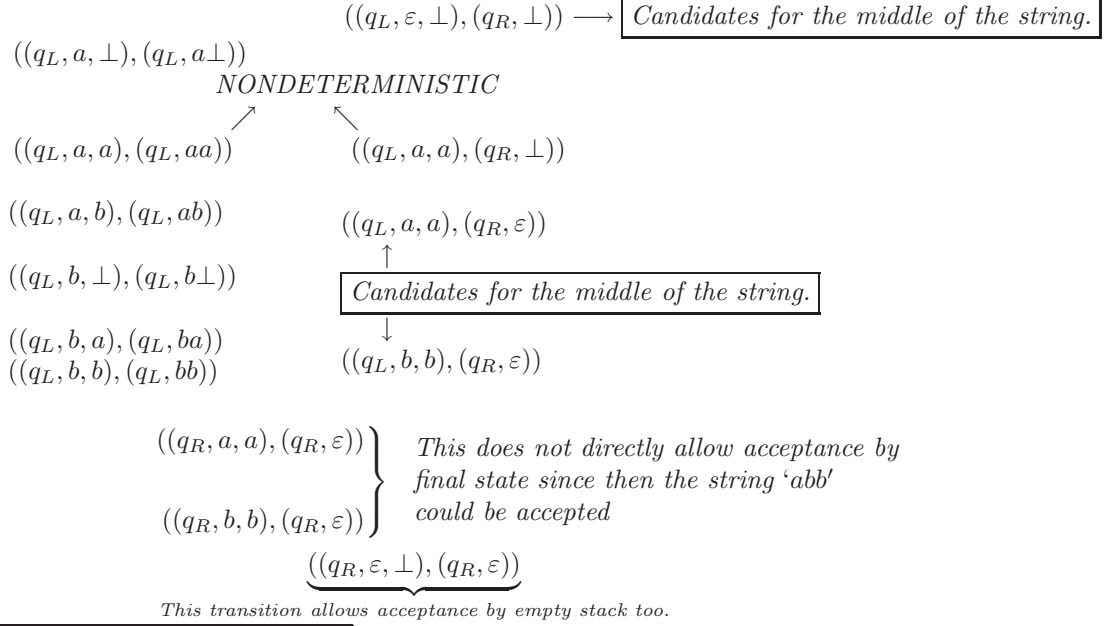
Example 3. $\{ww^r \mid w \in \{a, b\}^*\}$

$$\left. \begin{array}{l} Q = \{q_L^8, q_R\} \cup \{q_p^9\} \\ \Sigma = \{a, b\} \\ \Gamma = \{a, b, \perp\} \end{array} \right\} \begin{array}{l} \text{Acceptances} \\ \text{by empty} \\ \text{stack} \end{array}$$

Note

1. q_L is for start state.

2. q_p is for acceptance by final state.



Alternatively add a final state q_F
with the rule
 $((q_R, \varepsilon, \perp), (q_F, \perp)).$

Acceptances : E vs F

Note. When a $NPDA$ accepts by empty stack, then F is irrelevant and may safely be taken to be ϕ .

Theorem 36. *The two acceptances are equivalent i.e. for every N_E there exists a N_F with $\mathcal{L}_E(N_E) = \mathcal{L}_F(N_F)$ and vice-versa.*

Proof. Let $N = \langle Q, \Sigma, \Gamma, \delta, q_0, \perp, F \rangle$ be any $NPDA$ either E -type or F -type. Let s and t be two new states not in Q . and let \preccurlyeq be a new stack symbol. Now consider

$$N' = \langle Q \cup \{s, t\}, \Sigma, \Gamma \cup \{\preccurlyeq\}, \delta', s \perp, \{t\} \rangle$$

such that $\delta' = \delta \cup$

□

Claim 40. N' accepts the same language by both acceptances.

Proof. Outline

- (i) N' cannot reach its final state unless it has consumed all its input.
- (ii) It cannot empty its stack (i.e. \preccurlyeq cannot be popped) unless it first reaches its final state.
- (iii) Once it has reached its final state, it can only empty its stack.

Hence $\mathcal{L}_E = \mathcal{L}_F(N') = \mathcal{L}N'$.

□

Claim 41. $\mathcal{L}_E(N') \subseteq \mathcal{L}_E(N) = \mathcal{L}(N')$ if N is of type N_E
and $\mathcal{L}_E(N') \subseteq \mathcal{L}_F(N)$ if N is of type N_F .

Proof. Suppose $x \in \mathcal{L}(N')$. Then for some $n \geq 0$.

$$(s, x \preceq) \xrightarrow{N'} (q_0, x, \perp \preceq) \xrightarrow{N'}^n (q, y, \gamma, \preceq) \xrightarrow{N'} (t, y, \gamma \preceq) \xrightarrow{N'}^* (t, \varepsilon, \varepsilon).$$

Since $y = \varepsilon$

$$\underbrace{(q_0, x, \perp) \xrightarrow{N'}^n (q, \varepsilon, \gamma)}_{\downarrow}$$

where $q \in G$.

Now consider $(q, \varepsilon, \gamma \preceq) \xrightarrow{N'} (t, \varepsilon, \gamma \preceq)$

Case N by empty stack. Then $G = Q$ and $\Delta = \{\preceq\}$.

The transition $(q, \varepsilon, \gamma \preceq) \xrightarrow{N'} (t, \varepsilon, \gamma \preceq)$ implies $\gamma = \varepsilon$.

i.e. $(q, \varepsilon, \preceq) \xrightarrow{N'} (t, \varepsilon, \preceq)$.

$$\downarrow$$

$$(q_0, x, \preceq) \xrightarrow{N}^n (q, \varepsilon, \varepsilon). \text{ Hence } x \in \mathcal{L}_E(N).$$

Case N accepts by final state. Then $G = F$ and $\Delta = \Gamma \cup \{\preceq\}$.

Hence the transition $(q, \varepsilon, \gamma \preceq) \xrightarrow{N'} (t, \varepsilon, \gamma \preceq)$ implies $q \in F$.

Hence $(q_0, x, \perp) \xrightarrow{N}^n (q, \varepsilon, \gamma)$ where $q \in F$. Hence $x \in \mathcal{L}_F(N)$. □

Claim 42. $\mathcal{L}_E \subseteq \mathcal{L}(N')$ if $N = N_E$
and $\mathcal{L}_F \subseteq \mathcal{L}(N')$ if $N = N_F$

Proof. If $x \in \mathcal{L}_E(N)$ and $N = N_E$ then $(q_0, x, \perp) \xrightarrow{N}^* (q, \varepsilon, \varepsilon)$. Then it is easy to see that

$(s, x \preceq) \xrightarrow{N'} (q_0, x, \perp \preceq) \xrightarrow{N'}^* (q, \varepsilon, \preceq) \xrightarrow{N'} (t, \varepsilon, \preceq) \xrightarrow{N'} (t, \varepsilon, \varepsilon)$ and hence $x \in \mathcal{L}_E(N')$. On the other hand, if $x \in \mathcal{L}_F(N)$ and $N = N_F$ then $(q_0, x, \perp) \xrightarrow{N'}^* (q_F, \varepsilon, \gamma)$ and we may construct

$$(s, x, \preceq) \xrightarrow{N'} (q_0, x, \perp \preceq) \xrightarrow{N'}^* (q_F, \varepsilon, \gamma \preceq) \xrightarrow{N'} (t, \varepsilon, \gamma \preceq) \xrightarrow{N'}^* (t, \varepsilon, \varepsilon) \quad \square$$

NPDAs For CFLs

Theorem 37. For any, $A \subseteq \Sigma^*$, there exists a NPDA N such that $\mathcal{L}(N) = A$.

Proof. We first consider the case of $A^+ = A - \{\varepsilon\}$. We know there exists a grammar in CNF (Greibach Normal Form)

$G = \langle V, \Sigma, S, P \rangle$ where productions are all of the form

$$A \longrightarrow c B_1 B_2 \dots B_k$$

with $c \in \Sigma$ and $\{b_1, \dots, b_k\} \subseteq V$.

Let $N = \langle \{q_0\}, \Sigma, V, \delta, q_0 S, \phi^{10} \rangle$ where

$$\boxed{(q_0, \alpha) \in \delta(q_0, a, A) \quad \text{iff} \quad A \longrightarrow a\alpha \in P}$$

Claim 43. N simulates the leftmost derivations of G i.e.

$$S \xrightarrow{G}^* x\alpha \quad \text{iff} \quad (q_0, x, S) \xrightarrow{N}^* (q_0, \varepsilon, \alpha), \text{ for } 0000$$

¹⁰i.e. we show that $\mathcal{L}_E(N) = A^+$

(\Leftarrow) By induction on n where $(q_0, x, S) \xrightarrow[N]{n} (q_0, \delta, \alpha)$.

Basis $n = 0$. Then $x = \varepsilon$ and $s = \alpha$.

Ind Step. Let $x = ya$. Then clearly we have

$$(q_0, ya, S) \xrightarrow[N]{n-1} (q_0, a, \beta) \xrightarrow[N]{} (q_0, \varepsilon, \alpha)$$

from the $(n - 1)$ steps it is clear that

$$(q_0, y, S) \xrightarrow[N]{n-1} (q_0, \varepsilon, \beta)$$

Hence by the induction hypothesis we have $S \xRightarrow[G]{n-1} y\beta$. The move $(q_0, a, \beta) \xrightarrow[N]{} (q_0, \varepsilon, \alpha)$ is possible only if $\beta = A\gamma$ for some $A \in V$ and $\gamma \in V^*$ and further $\alpha = \eta\gamma$ for some $\eta \in V^*$. But from the construction of N we know that this is possible only if

$$\begin{array}{l} (q_0, \eta) \in \delta(q_0, a, A) \\ \text{iff } A \longrightarrow a\eta \in P. \end{array}$$

Hence $S \xRightarrow[G]{n-1} y\beta \xRightarrow[G]{} ya\eta\gamma = x\alpha$.

□

(\Rightarrow) Suppose $S \xRightarrow[G]{n} x\alpha$ by a leftmost derivation. Again by induction on n we show $(q_0, x, S) \xrightarrow[N]{n} (q_0, \varepsilon, \alpha)$

Ind Step. $x = ya$ and $S \xRightarrow[G]{n-1} yA\gamma \xRightarrow[G]{} ya\eta\gamma$: By the IH we

have $(q_0, y, S) \xrightarrow[N]{n-1} (q_0, \varepsilon, A\gamma)$ and hence

$$(q_0, ya, S) \xrightarrow[N]{n-1} (q_0, a, A\gamma) :$$

and $yA\gamma \xRightarrow[G]{} ya\eta\gamma$ is possible only if $A \longrightarrow a\eta \in P$. which implies $(q_0, \eta) \in \delta(q_0, a, A)$. Hence

$$(q_0, a, A\gamma) \xrightarrow[N]{} (q_0, \varepsilon, \eta\gamma) = (q_0, \varepsilon, \alpha).$$

The Case When $\varepsilon \in A$. The grammar G for A^+ is modified to include a new start symbol $S_0 \notin V$ with the productions $S_0 \longrightarrow S\varepsilon$. It is then clear that

$G_0 = \langle V_0, \Sigma, S_0, P_0 \rangle$ where $V_0 = V \cup \{S_0\}$, $P_0 = P \cup \{S_0 \longrightarrow S|\varepsilon\}$. has the property $\mathcal{L}(G_0) = A^+ \cup \{\varepsilon\} = A$.

The *NPDA*, N may then be modified N_0 with S_0 being the new stack marker and

$$\delta^0 = \delta\{((q_0, \varepsilon, S_0), (q_0, \varepsilon, S)), ((q_0, \varepsilon, S_0), (q_0, \varepsilon, \varepsilon))\}$$

such that $N_0 = \langle \{q_0\}, \Sigma, V_0, \delta_0, q_0, S_0, \phi \rangle$ with $\mathcal{L}_E(N_0) = \mathcal{L}_E(N) \cup \{\varepsilon\} = A^+ \cup \{\varepsilon\} = A$.

CFGs For NPDAs

The converse of the previous theorem viz. that the language accepted by a *NPDA* is a *CFL* is proven in two parts

Lemma 25. *The language E -accepted by a NPDA with a single state is a CFL*

↑
The Construction of the previous theorem
needs to be inverted

Proof. Let $N = \langle \{q_0\}, \Sigma, \Gamma, \delta, q_0, \perp, \phi \rangle$ be a NPDA. Construct $G = \langle \Gamma, \Sigma, \perp, P \rangle$ where for $c \in \Sigma \cup \{\varepsilon\}$, $B_1 \dots B_k \in \Gamma^*$

$$A \longrightarrow cB_1 \dots B_k \in P \quad \text{iff} \quad (q_0, B_1) \dots B_k \in \delta(q_0, c, A).$$

It is easy to see that G is CF and $\mathcal{L}(G) = \mathcal{L}_E(N)$. □

Lemma 26. *Every NPDA may be transformed into an equivalent one with a single state.*

★ *This implies that all state information may be maintained on the stack.*

Theorem 38. *The language accepted by any NPDA is context-free.*

Lemma 27. *Every NPDA may be simulated by a NPDA with a single state.*

Proof. WLOG from the construction of N' we may assume that the given NPDA N has a single final state.

$$N = \langle Q, \Sigma, \Gamma, \delta, s, \perp, \{t\} \rangle$$

and that N can empty its stack after entering this state.

Consider the machine N^\times (which accepts by empty stack) defined by

$$N^\times = \langle \{\star\}, \Sigma, \Gamma^\times, \delta^\times, \star, \langle s \perp t \rangle, \phi \rangle$$

where $\Gamma^\times = Q \times \Gamma \times Q$.

$$\begin{aligned} \delta^\times = \{ & ((\star, c, \langle pAq_k \rangle), (\star, \langle q_0B_1q_1 \rangle \langle q_1B_2q_2 \rangle \dots \langle q_{k-1}B_kq_k \rangle)) \\ & \mid ((p, c, A), (q_0, B_1B_2 \dots B_k)) \in \delta, \quad c \in \Sigma \cup \{\varepsilon\}, \\ & \text{and for all possible choices of } q_1, q_2, \dots, q_k \in Q \} \end{aligned}$$

Intuition N^\times simulates the behaviour of N by guessing nondeterministically what states N will pass through in the future and saves those guesses on the stack and verifies their correctness. □

Claim 44. $\forall n \geq 0 : (p, x, B_1B_2 \dots B_k) \xrightarrow[N]{n} (q, \varepsilon, \varepsilon) \Leftrightarrow$
 $\exists q_0, q_1, \dots, q_k \in Q : p = q_0 \wedge q = q_k \wedge$
 $(\star, x, \langle q_0B_1q_1 \rangle \langle q_1B_2q_2 \rangle \dots \langle q_{k-1}B_kq_k \rangle) \xrightarrow[N]{n} (\star, \varepsilon, \varepsilon)$

Proof. By induction on n .

Basis $n = 0$. $p = q$, $x = \varepsilon$ and $k = 0$ is trivial.

Ind Step. Assume $n > 0$ and the first step is such that

$$(p, x, B_1B_2 \dots B_k) \xrightarrow[N]{} (r, y, c_1 \dots c_m B_2 \dots B_k) \xrightarrow[N]{n} (q, \varepsilon, \varepsilon)$$

with $x = cy$ and $((p, c, B_1), (r, C_1 \dots C_m)) \in \delta$ and $c \in \Sigma \cup \{\varepsilon\}$.

By the IH there exists $r_0, r_1, \dots, r_{m-1}, q_1, \dots, q_{k-1}, q_k$ such that $r_0 = r$, $q = q_k$ and

$$(\dot{y}, \langle r_0 c_1 r_1 \rangle \langle r_1 c_2 r_2 \rangle \dots \langle r_{m-1} c_m q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots = \text{lan} q_{k-1} b_k q_k \xrightarrow{N^\times}^n (\star, \varepsilon, \varepsilon).$$

Also by the construction of N^\star

$$((\star, c, \langle p B_1 q_1 \rangle), (\star, \langle r_0 c_1 r_1 \rangle \langle r_1 c_2 r_2 \rangle \dots \langle r_{m-1} c_m q_1 \rangle)) \in \delta^\times$$

combing these we get

$$\begin{aligned} & (\star, x, \langle p B_1 q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots \langle q_{k-1} B_k q_k \rangle) \\ \xrightarrow{N^\times}^1 & (\star, y, \langle r_0 c_1 r_1 \rangle \langle r_1 c_2 r_2 \rangle \dots \langle r_{m-1} c_m q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots \langle q_{k-1} b_k q_k \rangle) \\ \xrightarrow{N^\times}^1 & (\star, \varepsilon, \varepsilon). \end{aligned}$$

$$(\Leftarrow) \text{ Suppose } (\star, x, \langle q_0 B_1 q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots \langle q_{k-1} B_k q_k \rangle) \xrightarrow{N^\times}^n (\star, \varepsilon, \varepsilon)$$

and let $((\star, c, \langle q_0 B_1 q_1 \rangle), (\star, \langle r_0 c_1 r_1 \rangle \langle r_1 c_2 r_2 \rangle \dots \langle r_{m-1} c_m q_1 \rangle))$ be the first transition applied where $c \in \Sigma\{\varepsilon\}$ and $m \geq 0$.

Then $x = cy$ for some y and

$$\begin{aligned} & (\star, x, \langle q_0 B_1 q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots \langle q_{k-1} B_k q_k \rangle) \\ \xrightarrow{N^\times} & (\star, y, \langle r_0 c_1 r_1 \rangle \langle r_1 c_2 r_2 \rangle \dots \langle r_{m-1} c_m q_1 \rangle \langle q_1 B_2 q_2 \rangle \dots \langle q_{k-1} b_k q_k \rangle) \\ \xrightarrow{N^\times}^n & (\star, \varepsilon, \varepsilon). \end{aligned}$$

By the IH we have $(r_0, y, c_1 c_2 \dots c_m B_2 \dots B_k) \xrightarrow{N}^{n-1} (q_k, \varepsilon, \varepsilon)$ and also by the construction of N^\times we have $((q_0, c, B_1), (r_0, c_1 \dots c_m)) \in \delta$. Considering these we get

$$(q_0, x, B_1 \dots B_k) \xrightarrow{N} (r_0, y, C_1 \dots C_m B_2 \dots B_k) \xrightarrow{N}^{n-1} (q_k, \varepsilon, \varepsilon).$$

Proof of lemma. for all $x \in \Sigma^\star$

$$\begin{aligned} x \in \mathcal{L}_F(N^\star) & \Leftrightarrow (\star, x, \langle s \perp t \rangle) \xrightarrow{N^\times}^\star (\star, \varepsilon, \varepsilon) \\ & \Leftrightarrow (s, x, \perp) \xrightarrow{N}^\star (t, \varepsilon, \varepsilon) \\ & \Leftrightarrow x \in \mathcal{L}_E(N). \end{aligned}$$

□

DETERMINISTIC PDAs

Definition 29. A DPDA is a NPDA satisfying the following constraints

$$\boxed{\forall a \in \Sigma \cup \{\varepsilon\} : \forall A \in \Gamma : \forall q \in Q : |\delta(q, a, A)| \leq 1}$$

i.e. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma^\star$

(i) in other words δ is a partial function which may be undefined for certain triples in $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma$

(ii) $\delta(q, \varepsilon, A) = (q', \alpha) \Rightarrow \forall a \in \Sigma : \Rightarrow \delta(q, a, A)$ is undefined.

i.e. given any input symbol a there is at most choice whether to consumed it or not and if it is to be consumed there is exactly one next configuration.

Examples. The language $\{a^n b^n \mid n \geq 0\}$ has a DPDA
 $\{w \in \{a, b\}^\star \mid \#a(w) = \#b(w)\}$ has a DPDA.

Some of our other examples e.g. $\{w^r \mid w \in \{a, b\}^\star\}$ have non-deterministic transitions which decide whether the midpoint has been reached.

However $\boxed{wcw^R \mid w \in \{a,b\}^*}$ has a deterministic PDA since the midpoint is clearly marked by a different symbol.

THE POWER OF DPDAs

Example. The language $A_2 = \{a^n b^{2n} \mid n > 0\}$ is accepted by a DPDA

$$\begin{aligned}
 Q_2 &= \{q_a, q_b\}, \Sigma = \{a, b\}, \Gamma_2 = \{B, \perp\}, F_2 = \phi \\
 \delta_2 : (q_a, a, \perp) &\mapsto (q_a, BB\perp) & (q_a, b, B) &\mapsto (q_b, \varepsilon) \\
 (q_a, a, B) &\mapsto (q_a, BBB) & (q_b, b, B) &\mapsto (q_b, \varepsilon) \\
 & & (q_b, \varepsilon, \perp) &\mapsto (q_b, \varepsilon, \varepsilon).0000
 \end{aligned}$$

Example. Similarly the language $A_1 = \{a^n b^n \mid n > 0\}$ has a DPDA.

The construction is exactly as above except that δ_1

$$\begin{aligned}
 \delta_1 : (q_a, a, \perp) &\mapsto (q_a, B\perp) & (q_a, b, B) &\mapsto (q_b, \varepsilon) \\
 (q_a, a, B) &\mapsto (q_a, BB) & (q_b, b, B) &\mapsto (q_b, \varepsilon) \\
 & & (q_b, \varepsilon, \perp) &\mapsto (q_b, \varepsilon, \varepsilon).
 \end{aligned}$$

Note that by changing Σ to $\Sigma \cup \{c\}$ where c is a new terminal symbol, the basic construction of PDAs remains unchanged.

Clearly the language $A_1 \cup A_2$ has a PDA obtained by taking the union of $\delta_1 \cup \delta_2$. However this PDA is nondeterministic because of the first two rules. But we can go further.

Theorem 39. $A_1 \cup A_2$ is not accepted by any DPDA.

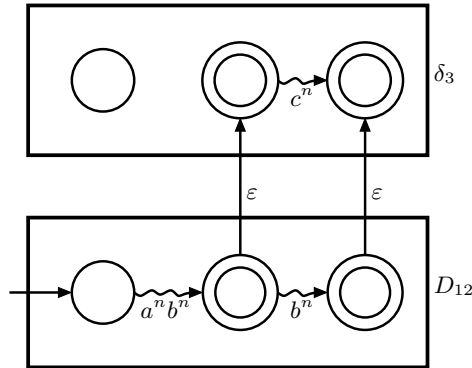
Proof. Assume there exists a DPDA $D_{12} = \langle Q_{12}, \Sigma, \Gamma, \delta_{12}, q_0, \perp, F_{12} \rangle$ accepting $A_1 \cup A_2$. □

Claim 45. There exists a NPDA which accepts the language $A_1 \cup A_{\text{@}} \cup A_3$ where $A_3 = \{a^n b^n c^n \mid n > 0\}$.

Proof. Let $N_{123} = \langle Q_{123}, \Sigma, \Gamma, \delta_{123}, q_0, \perp, F_{123} \rangle$

$$\begin{aligned}
 \text{where } Q_{123} &= F_{12} \cup F_3 & \text{i.e. } Q_3 &= \{\widehat{q} \mid q \in Q_{12}\} \\
 F_{123} &= F_{12} \cup F_3 & \text{with } F_3 &= \{\widehat{q} \mid q \in F_{12}\} \\
 \delta_{123} &= \delta_{12} \cup \delta_3 & \text{where } \delta_3 &= \{((q_F, \varepsilon, A), (\widehat{q}_F, A) \mid q_F \in F_{12}, A \in \Gamma\} \\
 & & & \cup \{((\widehat{q}_i, c, A), (\widehat{q}_j, \alpha) \mid \alpha \in \Gamma^* \wedge \\
 & & & \delta_{12}(q_i, b, A) = (q_j, \alpha)\}
 \end{aligned}$$

□



We have for d_{12} to accept $a^n b^n, n > 0$

$(q_0, a^n b^n, \perp) \xrightarrow[D_{12}]^* (q_F, \varepsilon, \alpha)$ with $q_F \in F$ is unique and since D_{12} is deterministic, for $a^n b^{2n}$ we

have

$(q_0, a^n b^{2n}, \perp) \xrightarrow[D_{12}]^* (q_F, b^n, \alpha) \xrightarrow[D_{12}]^* (q_{F'}, \varepsilon, \alpha')$

with $q_{F'}$ which implies that for $a^n b^n c^n$ we have by construction

$(q_0, a^n b^n c^n, \perp) \xrightarrow[N_{123}]^* (q_F, c^n, \alpha) \xrightarrow[N_{123}]^* (\widehat{q_{F'}}, \varepsilon, \alpha')$ with $\widehat{q_{F'}} \in F_{123}$.

Hence N_{123} accepts A_{123}

But we know that A_{123} is not CF since A_3 is not CF . Hence the assumption that there is a $DPDA D_{12}$ which accepts A_{12} is wrong !