# CS310 Automata Theory – 2017-2018

Nutan Limaye

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

Module 2: Extensions of DFA/NFAs
February, 2018

## Module - II: Different models of computation

What do we plan to do in this module?

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

Chapter 18, from the text of Dexter Kozen

Finite state transducers, machines that read and input and output a string.

Lecture notes shared on Moodle

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

Any textbook among the three reference books.

## Module - II: Different models of computation

2DFA: Two-way deterministic finite state automata.

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

Input head moves left/right on this tape.

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

It does not go to the right of $\$$.

Can potentially get stuck in an infinite loop!

## Formal definition of 2DFA

**Definition**

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

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

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

The following conditions are forced:

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

## 2DFA: Two-way deterministic finite state automata

Example

$L_1 = \{w \in \Sigma^* \mid \text{the second last letter is } a\}$.

2DFA is best described by giving its $\delta$ function.

Assume that initially the input head is on $\#$.

- Read the input and move right till \$ is encountered.
- Up on seeing \$ move left two positions.
- If that letter is a, then accept else reject.
- Handle other corner cases such as the word length is less than 2.

## 2DFA: Two-way deterministic finite state automata

Example

$L_1 = \{ w \in \Sigma^* \mid \text{the second last letter is } a \}$.

2DFA is best described by giving its $\delta$ function.

Assume that initially the input head is on $\#$.

The description of $\delta$ for the 2DFA for $L_1$ is given below.

$\delta(q_0, \#) = (q_1, R)$
$\delta(q_1, \$) = (q_{rej}, L)$
$\delta(q_1, c) = (q_2, R)$ for all $c \in \Sigma$
$\delta(q_2, \$) = (q_{rej}, L)$
$\delta(q_2, c) = (q_3, R)$ for all $c \in \Sigma$
$\delta(q_3, c) = (q_3, R)$ for all $c \in \Sigma$
$\delta(q_3, \$) = (q_4, L)$
$\delta(q_4, c) = (q_5, L)$ for all $c \in \Sigma$
$\delta(q_5, a) = (q_{acc}, L)$
$\delta(q_5, c) = (q_{rej}, L)$ for all $c \in (\Sigma \setminus \{a\})$.

## 2DFA: Two-way deterministic finite state automata

Examples

Let $\Sigma = \{a, b\}$ and $L$ be a regular language.
$L_2 = \{w \in \Sigma^* \mid w \cdot w \in L\}$

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA for $L$. The 2DFA for $L_2$ works as follows:

2DFA keeps two copies of states of $A$, say $Q^1$ and $Q^2$. Additionally it has an extra start state $q_0'$ and $|Q|$ many special state $q_{\leftarrow,i}$ for each $1 \leq i \leq |Q|$.

The first copy $Q^1$ is used to read the whole input the first time and do as per $\delta$ in that copy.Suppose we reach state $q_i^1$ at the end.

The special state $q_{\leftarrow,i}$ is used to move left after having read the input once.

The second copy $Q^2$ is used to read the input the second time.

## 2DFA: Two-way deterministic finite state automata

Examples

Let $\Sigma = \{a, b\}$ and $L$ be a regular language.
$L_2 = \{w \in \Sigma^* \mid w \cdot w \in L\}$

Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA for $L$. The 2DFA for $L_2$ works as follows:

Start from a special start state $q_0'$.
Reading $\#$ move to the state $q_0^1$ and move right.
Do exactly as per $\delta$ in $Q^1$ till \$ is encontered. Say the state reached is $q_i^1$ just before reading \$.
Upon seeing \$, move to a special state $q_{\leftarrow,i}$ and left.
In $q_{\leftarrow,i}$, reading any letter (other than $\#$), stay in $q_{\leftarrow,i}$ and move left.
In $q_{\leftarrow,i}$, reading $\#$, move to the state $q_i^2$ and move right.
Do exactly as per $\delta$ in $Q^2$ till \$ is encontered.
If the state is $q_f^2$, where $q_f \in F$, when reading \$ then go to state $q_{acc}$ and move left, else go to state $q_{rej}$ and move right.

## 2DFA: Two-way deterministic finite state automata

Examples

Let $\Sigma = \{a, b\}$ and $L$ be a regular language.

$L_1 = \{w \in \Sigma^* \mid \text{second letter from the end if } a\}$.

$L_2 = \{w \in \Sigma^* \mid w \cdot w \in L\}$

## Acceptance by 2DFA

**Definition**

Let $A$ be a 2DFA.

A word $w$ is said to be accepted by $A$ if $A$ reaches $q_{\text{acc}}$ on $w$.

A word $w$ is said to be rejected by $A$ if $A$ reaches $q_{\text{rej}}$ on $w$.

$A$ is said to accept a language $L$ if $\forall w \in L$, $A$ reaches $q_{\text{acc}}$.

2DFA may loop forever if $w \notin L$ or may enter $q_{\text{rej}}$.

**Lemma**

*If $L$ is regular then there is a 2DFA accepting $L$.*

This holds trivially.

## Languages accepted by 2DFA

Example

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

Can a 2DFA accept $L_{a,b}$?

$PAL = \{w \cdot w^R \mid w \in \{a,b\}^*\}$.

Can a 2DFA accept $PAL$?

## Power of 2DFAs

**Lemma**

*The class of language recognized by 2DFAs is regular.*

**Proof idea.**

For any language accepted by a 2DFA we will define a Myhill-Nerode relation.

How should we form word equivalences?

Using the behaviour of the input head for the given set of words.

To obtain finite index property, the equivalence should be with respect to states.

□

## Power of 2DFAs

**Lemma**

*The class of language recognized by 2DFAs is regular.*

**Proof.**

Let $T_x : Q \times \{\bowtie\} \to Q \times \{\bot\}$, which is defined as follows:

$T_x(p) := q$ if whenever $A$ enters $x$ on $p$ it leaves $x$ on $q$.

$T_x(\bowtie) := q$ $q$ is the state in which $A$ emerges on $x$ the first time.

$T_x(q) := \bot$ if $A$ loops on $x$ forever.

We will say that two words, $x, y$ are equivalent, i.e. $x \sim y$, if $T_x = T_y$.

We will show that $\sim$ is a Myhill-Nerode relation.

$\square$

## Power of 2DFAs

**Lemma**

*The class of language recognized by 2DFAs is regular.*

**Proof.**

Let $T_x : Q \times \{\bowtie\} \to Q \times \{\bot\}$, which is defined as follows:

| | | |
|---|---|---|
| $T_x(p) :=$ | $q$ | if whenever $A$ enters $x$ on $p$ it leaves $x$ on $q$. |
| $T_x(\bowtie) :=$ | $q$ | $q$ is the state in which $A$ emerges on $x$ the first time. |
| $T_x(q) :=$ | $\bot$ | if $A$ loops on $x$ forever. |

Total number of functions of the type

$T_x \leq (|Q| + 1)^{(|Q|+1)}$

$T_x = T_y \Rightarrow \forall z(xz \in F \Leftrightarrow yz \in F)$. Prove this.

$T_x = T_y \Leftrightarrow x \equiv_A y$

## Finite state transducer FST

A finite state trasducer is a finite state machine that reads an input and produces an output.

- the input head moves from left to right only.
- the input comes from some alphabet, say $\Sigma$ and the output may come from a possibly different alphabet, say $\Gamma$.
- after reading each latter, the machine may output a string from $\Gamma^*$.
- the output depends on the state of the machine and the letter being read at that state.

## Finite state transducer, FST

A finite state trasducer is a finite state machine that reads an input and produces an output.

**Definition**

A finite state transducer $T = (Q, \Sigma, \Gamma, \delta, q_0)$, where

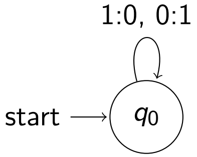
|  |  |  |  |
|---|---|---|---|
| $Q$ : | set of states, | $\Sigma$: | input alphabet |
| $\Gamma$: | output alphabet | $q_0$: | start state |
| $\delta$: | $Q \times \Sigma \to Q \times \Gamma^*$ |  |  |

## Finite state transducer, FST

### Example

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

Output: a string $w' \in \{0,1\}^*$ such that, $\#_0(w) = \#_1(w') = |w| - \#_1(w)$

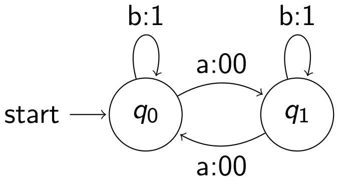


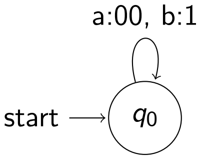
## Finite state transducer, FST

### Example

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

Output: a string $w' \in \{0, 1\}^*$ such that, if $\#_0(w') = 2 \cdot \#_a(w)$ and $\#_1(w') = \#_b(w)$





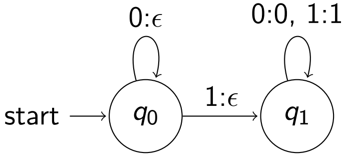
## Finite state transducer, FST

### Example

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

Output: $w' \in \{0,1\}^*$ such that, $\text{bin}(w') = \text{bin}(w) \pmod{2^i}$, where $2^i \le \text{bin}(w) < 2^{i+1}$

That is, for instance if $w = 0110$ then $w' = 10$ if $w = 1000110$ then $w' = 110$.


$$\begin{tikzcd}
\text{start} \arrow[r] & q_0 \arrow[loop above, "0{:}\epsilon"] \arrow[r, "1{:}\epsilon"] & q_1 \arrow[loop above, "0{:}0,\ 1{:}1"]
\end{tikzcd}$$

## Finite state transducer, FST

### Application

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

Output: $w' \in \{woof, yip\}^*$ such that, for each instance of the word 'bark'
for each instance of the word 'yelp' in $w$ output 'yip'
for all else do nothing.

Speech recognition literature uses such FSTs (and variants of FSTs) heavily to build tools.

## Output of an FST

Transition function $\delta$ extended to $\delta^*$.

**Definition**

Given an FST $T = (Q, \Sigma, \Gamma, q_0, \delta)$, let $\delta^* : Q \times \Sigma^* \to Q \times \Gamma^*$ be the function defined inductively as follows:

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

for any $q \in Q, w \in \Sigma^*$ and $a \in \Sigma$, $\delta^*(q, aw) = (q'', u \cdot u')$, where $\delta(q, a) = (q', u)$ and $\delta^*(q', w) = (q'', u')$.

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

## Function computed by an FST

Output of an FST

**Definition**

Let $T$ be an FST. The FST $T$ is said to output $u$ on $w$, denoted by $f_T(w) = u$, if there exists a state $q \in Q$ such that $\delta^*(q_0, w) = (q, u)$.

Functions computed by an FSTs.

**Definition**

A function $f : \Sigma^* \to \Gamma^*$ is said to be FST computable, if there exists an FST $T$ such that $\forall w \in \Sigma^*$, $f(w) = f_T(w)$.

## Compoſibility of FST computable functions

**Definition**

Let $f : \Sigma^* \rightarrow \Delta^*$ and $g : \Delta^* \rightarrow \Gamma^*$ be two functions. The function composition, $(g \circ f) : \Sigma^* \rightarrow \Gamma^*$, is defined as $(g \circ f)(w) = g(f(w))$.

**Lemma**

*Let $T_1 = (Q_1, \Sigma, \Delta, \delta_1, q_0^1)$ and $T_2 = (Q_2, \Delta, \Gamma, \delta_2, q_0^2)$ be two finite state transducers. There is a finite state transducer $T$ such that $f_T = (f_{T_2} \circ f_{T_1})$.*

**Proof.**

We define $T = (Q, \Sigma, \Gamma, \delta, q_0)$ using the product construction.

$Q = Q_1 \times Q_2$, $q_0 = (q_0^1, q_0^2)$.

$\delta((q_1, q_2), a) = ((q_1', q_2'), v)$, where

$\delta_1(q_1, a) = (q_1', w)$ and

$\delta_2^*(q_2, w) = (q_2', v)$.

$\square$

## Closure property of regular languages

**Lemma**

*Let $f_T : \Sigma^* \to \Gamma^*$ be an FST computable function computed by an FST $T$. Let $L \subseteq \Gamma^*$ be a regular language, then*

$$L' = f_T^{-1}(L) = \{w \in \Sigma^* \mid f(w) \in L\}$$

*is also regular.*

**Proof.**

Modified product construction. Let $A = (Q, \Gamma, \delta, q_0, F)$ be a DFA for $L$. Let $T = (Q_T, \Sigma, \Gamma, \delta_T, q_{(0,T)})$.

We design $B = (Q', \Sigma, \delta', q_0', F')$ a DFA for $L'$ as follows.

$Q' = Q \times Q_T$, $q_0' = (q_0, q_{(0,T)})$, $F' = F \times Q_T$,

$\delta'((p,q), a) = (\delta^*(p, w), r)$, where $\delta_T(q, a) = (r, w)$.

□

## Pushdown automata

NFA + Stack

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

$PAL = \{w \cdot w^R \mid w \in \Sigma^*\}$.

## Pushdown automata: formal definition

**Definition**

A non-deterministic pushdown automaton (NPDA) $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$, where

| | | | |
|---|---|---|---|
| $Q$: | set of states | $\Sigma$: | input alphabet |
| $\Gamma$: | stack alphabet | $q_0$: | start state |
| $\perp$: | start symbol | $F$: | set of final states |

$\delta \subseteq Q \times \Sigma \times \Gamma \times Q \times \Gamma^*$.

Understanding $\delta$

For $q \in Q, a \in \Sigma$ and $X \in \Gamma$, if $\delta(q, a, X) = (p, \gamma)$,
then $p$ is the new state and $\gamma$ replaces $X$ in the stack.

if $\gamma = \epsilon$ then $X$ is popped.
if $\gamma = X$ then $X$ stays unchanges on the top of the stack.
if $\gamma = \gamma_1\gamma_2 \ldots \gamma_k$ then $X$ is replaced by $\gamma_k$
and $\gamma_1\gamma_2 \ldots \gamma_{k-1}$ are pushed on top of that.

## Nondeterministic pushdown automata

Example: $L_{a,b} = \{a^n b^n \mid n \geq 0\}$.
$N = (Q = \{q_0, q_1, q_2, q'\}, \Sigma = \{a, b\}, \Gamma = \{B\}, q_0, F = \{q_2\})$.

$\delta(q_0, a, \perp) = (q_0, B\perp)$

$\delta(q_0, a, B) = (q_0, BB)$ *push $B$ while reading $a$ on $q_0$*

$\delta(q_0, b, B) = (q_1, \epsilon)$ *move to $q_1$ on seeing $b$*

$\delta(q_1, b, B) = (q_1, \epsilon)$ *pop $B$ while reading $b$ on $q_1$*

$\delta(q_1, b, \perp) = (q', \perp)$ *extra $b$ in the input then go to some state $q'$*

$\delta(q_1, \epsilon, \perp) = (q_2, \perp)$ *else go to accepting state $q_2$*

## Nondeterministic pushdown automata

Example: $PAL = \{w \cdot w^R \mid w \in \Sigma^*\}$.
$N = (Q = \{q_0, q_1, q_2, q'\}, \Sigma = \{a, b\}, \Gamma = \{A, B\}, q_0, F = \{q_2\})$.

$$\delta(q_0, a, \perp) = \{(q_0, A\perp), (q_1, A\perp)\}$$
$$\delta(q_0, b, \perp) = \{(q_0, B\perp), (q_1, B\perp)\}$$
$$\delta(q_0, a, A) = \{(q_0, AA), (q_1, AA)\},\ \delta(q_0, a, B) = \{(q_0, AB), (q_1, AB)\}$$
$$\delta(q_0, b, A) = \{(q_0, BA), (q_1, BA)\},\ \delta(q_0, b, B) = \{(q_0, BB), (q_1, BB)\}$$
$$\delta(q_1, a, A) = (q_1, \epsilon)$$
$$\delta(q_1, b, B) = (q_1, \epsilon)$$
$$\delta(q_1, a, B) = (q', \epsilon)$$
$$\delta(q_1, b, A) = (q', \epsilon)$$
$$\delta(q_1, a, \perp) = (q', \epsilon)$$
$$\delta(q_1, b, \perp) = (q', \epsilon)$$
$$\delta(q_1, \epsilon, \perp) = (q_2, \perp)$$

## Configuration of an NPDA

### Definition (Configurations)

A configuration of an NPDA $A = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ is a three tuple $(q, w, \gamma)$, where $q \in Q$, $w \in \Sigma^*$, and $\gamma \in \Gamma^*$.

if $(p, \gamma) \in \delta(q, a, X)$ then $\forall w \in \Sigma^*$ and $\gamma' \in \Gamma^*$,

$$(q, a \cdot w, X\gamma') \;\vdash\; (p, w, \gamma \cdot \gamma')$$

Let $I, J$ are two configurations of $A$.

We say that $I \vdash^k J$ iff $\exists I'$ such that $I \vdash I'$ and $I' \vdash^{k-1} J$.

## Language recognized by pushdown automata

**Definition**

We say that a word is accepted by an NPDA $A$ if $(q_0, w, \perp) \vdash^* (q, \epsilon, \gamma)$, where $q \in F$. acceptance by a final state.

A language $L$ is said to be recognized by an NPDA $A$ if the set $\{w \mid w \text{ is accepted by } A\}$ is the same as $L$.

The class of languages recognized by NPDAs is called Context-free languages.

Another notion of acceptance of words:

We say that a word is accepted by an NPDA $A$ if $(q_0, w, \perp) \vdash^* (q, \epsilon, \epsilon)$, where $q \in Q$. acceptance by an empty stack.

## Context-free languages

Examples

$$\text{PAL} = \{w \cdot w^R \mid w \in \Sigma^*\}.$$

$$\text{Balanced} = \{w \in \{(,),[,]\} \mid w \text{ balanced string of paranthesis } \}.$$

$$L_{a/b/c} = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}.$$

## Closure property of CFLs

**Lemma**

*Let $L_1, L_2$ be two context-free languages. Then $L_1 \cup L_2$ is also a context-free language.*

**Proof.**

Let $L_1$ be accepted by an NPDA $N_1 = (Q^1, \Sigma, \Gamma, \delta^1, q_0^1, \perp, F^1)$.
Let $L_2$ be accepted by an NPDA $N_2 = (Q^2, \Sigma, \Gamma, \delta^2, q_0^2, \perp, F^2)$.
Let $N = (Q, \Sigma, \Gamma, \delta, q_0, \perp, F)$ be an NPDA defined as follows:

$Q = Q^1 \cup Q^2 \cup \{q_0\}$, $F = F^1 \cup F^2$,
$\delta = \delta_1 \cup \delta_2 \cup \{\delta(q_0, \epsilon, \perp) = \{(q_0^1, \perp), (q_0^2, \perp)\}\}$.

$\square$

## Context-free languages

Examples

$\mathrm{PAL} = \{w \cdot w^R \mid w \in \Sigma^*\}.$

$\text{Balanced} = \{w \in \{(,),[,]\} \mid w \text{ balanced string of paranthesis }\}.$

$L_{a/b/c} = \{a^i b^j c^k \mid i \neq j \text{ or } j \neq k\}.$

$L_{a/b/c} = \{a^i b^j c^k \mid i \neq j \text{ and } j \neq k\}.$ ?

$L_{a,b,c} = \{a^n \cdot b^n \cdot c^n \mid n \geq 0\}.$ ?

$\mathrm{EQ} = \{w \cdot w \mid w \in \Sigma^*\}.$ ?

## Non-context-free languages

**Lemma (Pumping lemma for CFLs)**

*Say $L$ is a language over the alphabet $\Sigma^*$. If*

- ☹ *for all $n \in \mathbb{N}$,*
- ☺ *$\exists z \in \Sigma^*$, such that $z \in L$*
- ☹ *for all possible ways of breaking $z$ into $z = u \cdot v \cdot w \cdot x \cdot y$, s.t. $|v \cdot w \cdot x| \leq n$ and $|v \cdot x| > 0$,*
- ☺ *$\exists i \in \mathbb{N}$ s. t. $u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$,*

*then $L$ is not a CFL.*

## Applications of the pumping lemma for CFLs

Let $L_{a,b,c} = \{a^m b^m c^m \mid m \geq 0\}$

- ☹ For any chosen $n$,
- ☺ let $z = a^n \cdot b^n \cdot c^n$
- ☹ For any split of $z$ into $u, v, w, x, y$
- ☺ as $|v \cdot w \cdot x| \leq n$
  Either $v \cdot w \cdot x$ has no c's, or no a's.
  Therefore, $u \cdot v^0 \cdot w \cdot x^0 \cdot y \notin L$.

Say $L$ is a language over the alphabet $\Sigma^*$. If

- ☹ for all $n \in \mathbb{N}$,
- ☺ $\exists z \in \Sigma^*$, such that $z \in L$
- ☹ for all possible ways of breaking $z$ into $z = u \cdot v \cdot w \cdot x \cdot y$, s.t. $|v \cdot w \cdot x| \leq n$ and $|v \cdot x| > 0$,
- ☺ $\exists i \in \mathbb{N}$ s. t. $u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$,

then $L$ is not a CFL.

## Applications of the pumping lemma for CFLs

Let $EQ = \{w \cdot w \mid w \in \{a, b\}^*\}$.

- ☹ For any chosen $n$,
- ☺ let $z = a^n \cdot b^{2n} \cdot a^n \cdot b^{2n}$
- ☹ For any split of $z$ into $u, v, w, x, y$
- ☺ Note that $|v \cdot w \cdot x| \le n$.
  (after some case analysis.)
  Therefore, $u \cdot v^0 \cdot w \cdot x^0 \cdot y \notin L$.

Say $L$ is a language over the alphabet $\Sigma^*$. If

- ☹ for all $n \in \mathbb{N}$,
- ☺ $\exists z \in \Sigma^*$, such that $z \in L$
- ☹ for all possible ways of breaking $z$ into $z = u \cdot v \cdot w \cdot x \cdot y$, s.t. $|v \cdot w \cdot x| \le n$ and $|v \cdot x| > 0$,
- ☺ $\exists i \in \mathbb{N}$ s. t. $u \cdot v^i \cdot w \cdot x^i \cdot y \notin L$,

then $L$ is not a CFL.

## Context-free grammars

Inductive definition of PAL.

$\epsilon$ is in PAL.

If $w$ is in PAL then $0 \cdot w \cdot 0 \in$ PAL.

If $w$ is in PAL then $1 \cdot w \cdot 1 \in$ PAL.

Context-free grammar for PAL.

$S \to \epsilon$.

$S \to 0S0$.

$S \to 1S1$.

## Context-free grammars

PAL' = $\left\{ w \cdot c \cdot w^R \mid w \in \{0,1\}^*, c \in \{0,1,\epsilon\} \right\}$
Inductive definition of PAL'.

$\epsilon, 0, 1$ is in PAL'.

If $w$ is in PAL' then $0 \cdot w \cdot 0 \in$ PAL'.

If $w$ is in PAL' then $1 \cdot w \cdot 1 \in$ PAL'.

Context-free grammar for PAL'.

$S \to \epsilon$.

$S \to 0$.

$S \to 1$.

$S \to 0S0$.

$S \to 1S1$.

## Context-free grammar

**Definition**

A context-free grammar (CFG) $G$ is given by $(V, T, P, S_0)$, where

$V$ is a set of variables,

$T$ is a set of terminal symbols or the alphabet,

$P$ is a set of productions, $P \subseteq V \times (V \cup T)^*$,

$S_0 \in V$, a start symbol.

Example: Grammar for PAL.

$S \to \epsilon.$

$S \to 0.$

$S \to 1.$

$S \to 0S0.$

$S \to 1S1.$

$G_{pal} = (V, T, P, S_0)$ such that

$V = \{S\}$,

$T = \{0, 1\}$,

$P = \{S \to \epsilon, S \to 0, S \to 1, S \to 0S0, S \to 1S1\}$,

$S_0 = S$.

## Context-free grammars

Example: $L_{a,b} = \{a^n b^n \mid n \geq 0\}$

$$S \rightarrow \epsilon \mid aSb$$

Example: $L' = \{a^i b^j \mid i < j\}$

$$S' \rightarrow b \mid aS'b \mid S'b$$

Example: $L'' = \{a^i b^j \mid i > j\}$

$$S'' \rightarrow a \mid aS''b \mid aS''$$

Example: $L = \{a^i b^j \mid i \neq j\}$

$$S \rightarrow S' \mid S''$$
$$S' \rightarrow b \mid aS'b \mid S'b$$
$$S'' \rightarrow a \mid aS''b \mid aS''.$$

## Derivations of a CFG

**Definition**

Let $G$ be a CFG given by $(V, T, P, S_0)$.

Let $w, w' \in (V \cup T)^*$,

let $A \in V$ and let $(A \to v) \in P$ be a production in the grammar, where $v \in (V \cup T)^*$.

Then we say that $w \cdot A \cdot w'$ **derives** $w \cdot v \cdot w'$ **in one step**.

We denote it as follows: $w \cdot A \cdot w' \Rightarrow w \cdot v \cdot w'$.

**Definition ($\Rightarrow^*$)**

Let $G$ be a CFG given by $(V, T, P, S_0)$.

For all $\alpha \in (V \cup T)^*$, we say that $\alpha \Rightarrow^0 \alpha$.

For all $\alpha, \beta, \gamma \in (V \cup T)^*$,

if $\alpha \Rightarrow^{k-1} \beta$ and $\beta \Rightarrow \gamma$ then $\alpha \Rightarrow^k \gamma$.

For all $\alpha, \beta \in (V \cup T)^*$, we say that $\alpha \Rightarrow^* \beta$, if $\exists k \geq 0$ s.t. $\alpha \Rightarrow^k \beta$.

## Language of a CFG

**Definition**

Let $G$ be a CFG given by $(V, T, P, S_0)$. The **language of** $G$, $L(G)$, is the set of all the strings over $T$ which can be derived from $S_0$, i.e.

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

**Lemma**

*$L(G_{pal})$ is equal to PAL.*
*$\forall w \in \{0,1\},^* \ w \in PAL$ if and only if $w = w^R$.*

**Proof.**

By Induction on $|w|$. □

## Examples

Give context-free grammars for the following languages.

$\{a^i \cdot b^j \cdot c^k \mid \text{ either } i \neq j \text{ or } j \neq k\}$.

First we give a grammar for $\{a^i \cdot b^j \cdot c^k \mid i \neq j\}$.

$S \to S'C \mid S''C$
$S' \to b \mid aS'b \mid S'b$
$S'' \to a \mid aS''b \mid aS''$.
$C \to cC \mid \epsilon$

Now we give a grammar for $\{a^i \cdot b^j \cdot c^k \mid j \neq k\}$.

$S \to AS' \mid AS''$
$S' \to c \mid bS'c \mid S'c$
$S'' \to b \mid bS''c \mid bS''$.
$C \to aA \mid \epsilon$

Now simply take the union.

## Chomsky normal form

**Definition**

A context-free grammar is said to be in Chomsky normal form if every rule is of the form

$$A \to BC$$

$$A \to a$$

where $a \in T$, $A, B, C \in V$, neither $B$ nor $C$ is the start variable, i.e. start variabe does not appear on the right of any rule. Moreover, epsilon does not appear on the right of any rule except as $S \to \epsilon$.

**Lemma**

*Any context-free grammar $G$ can be converted into another context-free grammar $G'$ such that $L(G) = L(G')$ and $G'$ is in the Chomsky normal form.*

## Chomsky normal form

Step 1: Add a new start symbol

$S_0 \to S$

Step 2: Remove $\epsilon$ rules.

Suppose $A \to \epsilon$ is a rule and $A$ is not the start symbol.

If $R \to uAv$ is a rule then delete the rule and add $R \to uv$ to the rules.

If $R \to uAvAw$ is a rule then delete the rule and add $R \to uvAw \mid uAvw \mid uvw$.

If $R \to A$ is a rule then delete the rule and add $R \to \epsilon$ unless $R \to \epsilon$ was already removed.

Step 3: Remove unit rules.

If $A \to B$ is a rule and if $B \to u$ appears

then remove $A \to B$ and add $A \to u$.

## Chomsky normal form

Step 1: Add a new start symbol

Step 2: Remove $\epsilon$ rules.

Step 3: Remove unit rules.

Step 4: Put the rest of the rules in the proper form.

If $A \to u_1 u_2 \dots u_k$ is a rule, where $k \geq 3$ and $u_i \in V \cup T$

Remove this rule and add the following rules:

$A \to u_1 A_1$, $A_1 \to u_2 A_2 \dots A_{k-2} \to u_{k-1} u_k$.

If $u_i \in T$, moreover replace each $u_i$ with a variable $U_i$ and add $U_i \to u_i$.

## Equivalence of CFGs and PDAs

**Theorem**

*A language is context-free if and only if it is generated by a context-free grammar.*

Proof idea: from CFGs to NPDAs.

Assume CFG is in the Chomsky normal form.

Push $S_0$ on the stack and make it the current variable.

Push non-deterministically one of the strings in the right hand side of the rule generated from the current variable on the stack.

e.g. $A \to BC \mid DE$ then non-deterministically choose either $BC$ or $DE$ and depending on the choice, say it is $BC$, push the string $BC$ on the stack with $B$ on the top of the stack.

If the the top is a terminal, then match it off with the input bit,

if the top of the stack is $\perp$ then accept

else make that the new current variable,

Repeat the above procedure. (It will either accept or loop forever.)