

Computational Models - Lecture 6¹

Handout Mode

Iftach Haitner.

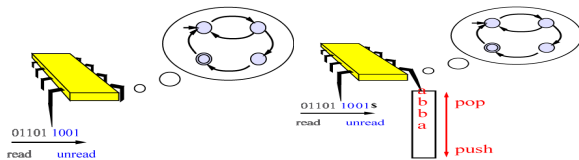
Tel Aviv University.

December 05, 2016

¹Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University.

Outline

- ▶ Push Down Automata (PDA)
- ▶ Closure properties for CFL and testing properties.
- ▶ **Equivalence** of CFGs and PDAs



- ▶ Sipser's book, [2.1](#), [2.2](#)&[2.3](#)

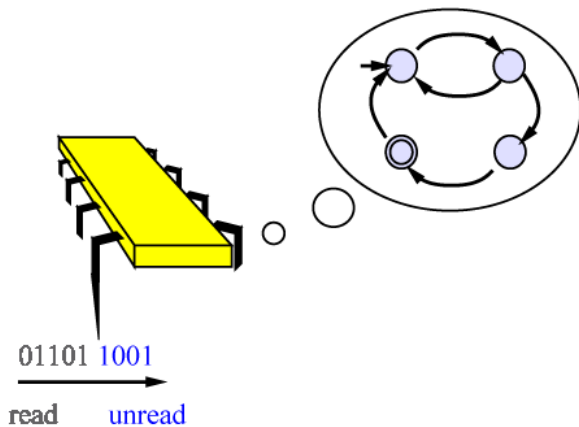
Part I

Push-Down Automata

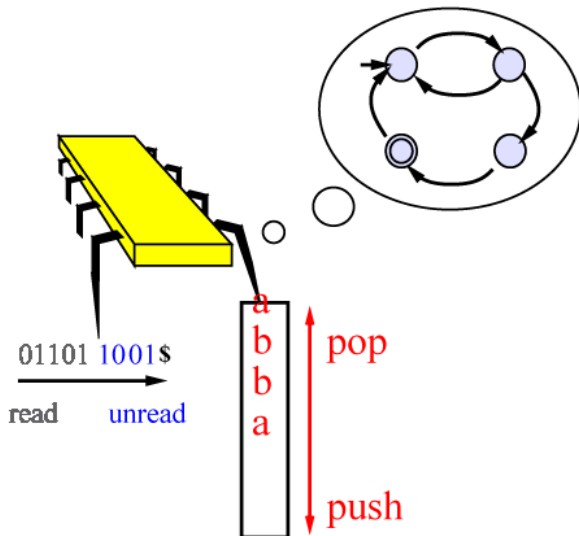
String generators and string acceptors

- ▶ Regular expressions are string **generators** – they tell us how to generate all strings in a language \mathcal{L}
- ▶ Finite automata (DFA, NFA) are string **acceptors** – they tell us if a specific string w is in \mathcal{L}
- ▶ CFGs are string **generators**
- ▶ Are there string **acceptors** for CFLs?
- ▶ YES! **Push-down automata**

Finite automaton



PushDown automaton



(ignore the '\$' sign)

Example 1 — PDA for $\mathcal{L}_1 = \{0^n 1^n : n \geq 0\}$

Informally:

1. Read input symbols
 - 1.1 Push each read 0 on the stack
 - 1.2 Pop a 0 for each read 1
2. Accept if stack is empty after last symbol read, and no 0 appears after 1

Recall that \mathcal{L}_1 is not regular

Example 2 — PDA for $\mathcal{L}_2 = \{a^i b^j c^k : i = j \vee i = k\}$

Informally:

Read and push a 's

Either pop and match with b 's or pop and match with c 's

A non-deterministic choice

PDA – Formal definition

A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$, where

- ▶ Q is a finite set called the **states**,
- ▶ Σ is a finite set called the **input alphabet**,
- ▶ Γ is a finite set called the **stack alphabet**,
- ▶ $\delta: Q \times \Sigma_\epsilon \times \Gamma_\epsilon \mapsto \mathcal{P}(Q \times \Gamma_\epsilon)$ is the **transition function**,²
- ▶ $q_0 \in Q$ is the **starting state**, and
- ▶ $F \subseteq Q$ is the set of **accepting states**.

² $X_\epsilon := X \cup \{\epsilon\}$.

Formal model of computation

Recall that for $a \in (X_\varepsilon)^*$, we let $d(a) \in X^*$ be a without the ε symbols.

Definition 1 ($\tilde{\delta}$)

For PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, define $\tilde{\delta}_P: Q \times \Sigma_\varepsilon \times \Gamma^* \mapsto \mathcal{P}(Q \times \Gamma^*)$ by:
 $(q', s') \in \tilde{\delta}_P(q, \sigma, s)$, if $\exists a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$ s.t.:

- ▶ $(q', b) \in \delta(q, \sigma, a)$
- ▶ $s = d(at)$ and $s' = d(bt)$

Namely, reading the symbol σ from the input, can change *configuration* (state and stack value) (q, s) into configuration (q', s') .

It might be that

- ▶ $|s'| = |s|$: $a, b \neq \varepsilon$ or $a = b = \varepsilon$
- ▶ $|s'| > |s|$: $a = \varepsilon$ and $b \neq \varepsilon$
- ▶ $|s'| < |s|$: $a \neq \varepsilon$ and $b = \varepsilon$

As usual, we omit the subscript P when clear from the context.

Formal model of computation, cont.

Definition 2

A PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** $w \in \Sigma^*$, if exist $a_1, \dots, a_k \in \Sigma_\varepsilon$, $r_0, \dots, r_k \in Q$ and $s_0, \dots, s_k \in \Gamma^*$, s.t.

- ▶ $w = d(a_1, \dots, a_k)$.
- ▶ $r_0 = q_0$ and $r_k \in F$.
- ▶ $s_0 = \varepsilon$.
- ▶ $(r_{i+1}, s_{i+1}) \in \tilde{\delta}_P(r_i, a_{i+1}, s_i)$, for all $0 \leq i < k$.

Definition 3

The language **accepted** by PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, denoted $\mathcal{L}(P)$, is the set of all strings $w \in \Sigma^*$ accepted by P .

An equivalent definition

Definition 4

PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ **accepts** $w \in \Sigma^*$, if $\widehat{\delta}_P(q_0, w, \varepsilon) \cap (F \times \Gamma^*) \neq \emptyset$

Definition 5 (ε -environment)

For PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ and $(q, s) \in Q \times \Gamma^*$, let

$E(q, s) = \{(q', s') \in Q \times \Gamma^* : \exists (q_1, s_1), \dots, (q_k, s_k) \text{ s.t.}$

$(q_1, s_1) = (q, s) \wedge (q_k, s_k) = (q', s') \wedge \forall i \in [k-1] : (q_{i+1}, s_{i+1}) \in \widetilde{\delta}(q_i, \varepsilon, s_i)\}$.

$E(\mathcal{A} \subseteq Q \times \Gamma^*) = \bigcup_{(q,s) \in \mathcal{A}} E(q, s)$

Definition 6 ($\widehat{\delta}$)

For PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, define $\widehat{\delta}_P: Q \times \Sigma^* \times \Gamma^* \mapsto \mathcal{P}(Q \times \Gamma^*)$ by:

$$\widehat{\delta}_P(q, w, s) = \begin{cases} E(q, s), & w = \varepsilon, \\ E\left(\bigcup_{(q', s') \in \widehat{\delta}_P(q, w_1, \dots, w_{n-1}, s)} \widetilde{\delta}_P(q', w_n, s')\right), & n = |w| \geq 1. \end{cases}$$

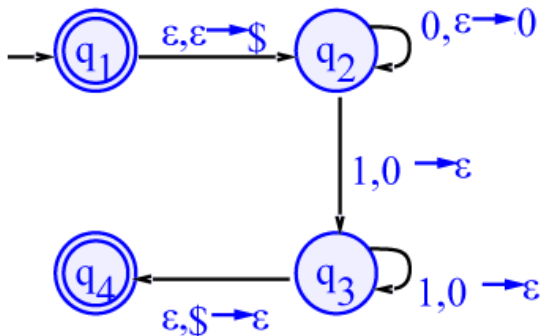
Diagram notation

When drawing the automata diagram, we use the following notation

- ▶ Transition $a, b \rightarrow c$ from state q to q' means $(q', c) \in \delta(q, a, b)$, and informally means the automata
 - ▶ read a from input
 - ▶ pop b from stack
 - ▶ push c onto stack

- ▶ Meaning of ε transitions ((informally):
 - ▶ $a = \varepsilon$: don't read input
 - ▶ $b = \varepsilon$: don't pop any symbol
 - ▶ $c = \varepsilon$: don't push any symbol

A PDA for $\mathcal{L}_1 = \{0^n 1^n : n \geq 0\}$



Claim 7

$0011 \in L(P)$.

Proof: take

	$w'_1 = \varepsilon$	$w'_2 = 0$	$w'_3 = 0$	$w'_4 = 1$	$w'_5 = 1$	$w'_6 = \varepsilon$
$s_0 = \varepsilon$	$s_1 = \$$	$s_2 = 0\$$	$s_3 = 00\$$	$s_4 = 0\$$	$s_5 = \$$	$s_6 = \varepsilon$
$r_0 = q_1$	$r_1 = q_2$	$r_2 = q_2$	$r_3 = q_2$	$r_4 = q_3$	$r_5 = q_3$	$r_6 = q_4$

A PDA for $\mathcal{L}_1 = \{0^n 1^n : n \geq 0\}$

We want to show that $L(P) = \mathcal{L}_1 = \{0^n 1^n : n \geq 0\}$

What do we need to prove?

Claim 8

- ▶ $\widehat{\delta}(q_1, \varepsilon, \varepsilon) = \{(q_1, \varepsilon), (q_2, \$)\}$.
- ▶ $\widehat{\delta}(q_1, 0^k, \varepsilon) = \{(q_2, 0^k \$)\}$, for $k \geq 1$.
- ▶ $\widehat{\delta}(q_1, 0^k 1^i, \varepsilon) = \{(q_3, 0^{k-i} \$)\}$, for $k > i \geq 1$.
- ▶ $\widehat{\delta}(q_1, 0^k 1^k, \varepsilon) = \{(q_3, \$), (q_4, \varepsilon)\}$, for $k \geq 1$.
- ▶ $\widehat{\delta}(q_1, w, \varepsilon) = \emptyset$, for $w \notin \{0^k 1^i \mid k \geq i \geq 0\}$.

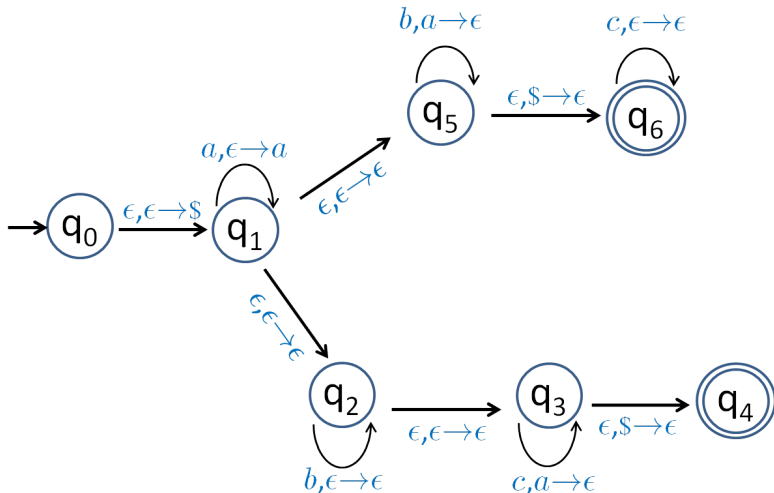
Knowing when stack is empty

It is convenient to be able to know when the stack is **empty**, but there is **no built-in mechanism** to do that.

Solution

1. Start by pushing **\$** onto stack.
2. When you see it again, stack is empty.

A PDA for $\mathcal{L}_2 = \{a^i b^j c^k : i = j \vee i = k\}$



A PDA for $\mathcal{L}_2 = \{a^i b^j c^k : i = j \vee i = k\}$, cont.

- ▶ Non-determinism is essential here!
- ▶ Unlike finite automata, non-determinism **does add power**.
- ▶ But we saw **deterministic** algorithm to decide any CFL (and as we see later, CFLs are exactly the languages decided by PDAs)!
- ▶ How to prove that non-determinism adds power?

⋮
- ▶ Does not seem trivial or immediate.
- ▶ Another example: $\mathcal{L} = \{x^n y^n : n \geq 0\} \cup \{x^n y^{2^n} : n \geq 0\}$ is accepted by a non-deterministic PDA, but **not** by a deterministic one. (Proof? Book!)

PDA languages

The Push-Down Automata Languages, \mathcal{L}_{PDA} , is the set of all languages that can be described by some PDA:

$$\triangleright \mathcal{L}_{\text{PDA}} = \{\mathcal{L}(M) : M \text{ is a PDA}\}$$

It is immediate that $\mathcal{L}_{\text{PDA}} \supsetneq \mathcal{L}_{\text{DFA}}$: every DFA is just a PDA that **ignores** the stack.

$$\triangleright \mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}} ?$$

$$\triangleright \mathcal{L}_{\text{PDA}} \subseteq \mathcal{L}_{\text{CFG}} ?$$

$$\triangleright \mathcal{L}_{\text{PDA}} = \mathcal{L}_{\text{CFG}} !!!$$

Proof in last hour of class.

Part II

Closure Properties of Context-Free Languages

Simple closure properties of CFL's

- ▶ CFL's are closed under
 - ▶ Union: $S \rightarrow S_1 \mid S_2$
 - ▶ Concatenation: $S \rightarrow S_1 S_2$
 - ▶ Star: $S_{new} \rightarrow \varepsilon \mid S_{old} \mid S_{old} S_{new}$
- ▶ What about complement and intersection?

Intersection

$$S_1 \rightarrow A_1 B_1$$

$$A_1 \rightarrow 0A_11|\varepsilon$$

$$B_1 \rightarrow 2B_1|\varepsilon$$

$$\mathcal{L}_1 = 0^n 1^n 2^*$$

$$S_2 \rightarrow A_2 B_2$$

$$A_2 \rightarrow 0A_2|\varepsilon$$

$$B_2 \rightarrow 1B_22|\varepsilon$$

$$\mathcal{L}_2 = 0^* 1^n 2^n$$

- ▶ $\mathcal{L}_1 \cap \mathcal{L}_2 = 0^n 1^n 2^n$
- ▶ \mathcal{L}_1 and \mathcal{L}_2 are CFLs (why?),
- ▶ But $\mathcal{L}_1 \cap \mathcal{L}_2$ is not a CFL.
- ▶ But can't we run two PDA's in parallel, and accept iff both accept??
- ▶ What about intersection of a CFL with a regular language?

When CFL Intersects Regular Language

- ▶ Are the context free languages closed under **intersection** with a **regular language**?
- ▶ That is, if \mathcal{L}_1 is context free languages, and \mathcal{L}_2 is regular, must $\mathcal{L}_1 \cap \mathcal{L}_2$ be context free languages?
- ▶ YES!
 - ▶ Run PDA \mathcal{L}_1 and DFA \mathcal{L}_2 “in parallel” (just like the intersection of two regular languages).
 - ▶ Formal details omitted (**but you should be able to figure them out**).

Applications

- ▶ Is $\mathcal{L} = \{w \in \{0, 1, 2\}^* : \#_0(w) = \#_1(w) = \#_2(w)\}$ context free?
 - ▶ $\mathcal{L} \cap 0^*1^*2^* = \{0^n1^n2^n : n \geq 0\}$ is **not** context free.
 - ▶ $0^*1^*2^*$ is regular.
 - ▶ Context free languages intersected with a **regular** languages **are** context free.
 - ▶ So \mathcal{L} is **not** a context free language
- ▶ This could also be established using pumping lemma, but proof above is more elegant.

Complementation

The fact that CFLs are **not** closed under **intersection**, but are closed under **union**, implies they are **not closed** under **complementation**, as

$$\mathcal{L}_1 \cap \mathcal{L}_2 = \overline{\overline{\mathcal{L}_1} \cup \overline{\mathcal{L}_2}}.$$

We give a simple example where \mathcal{L} is **not** CFL but $\overline{\mathcal{L}}$ is.

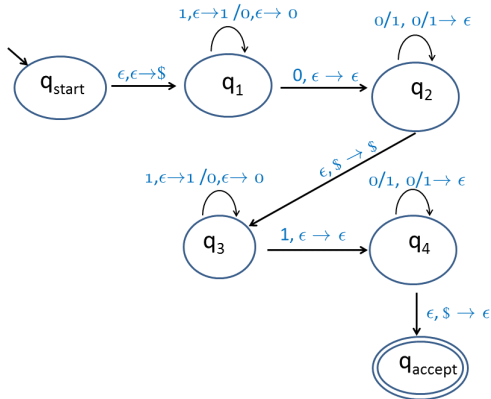
- ▶ Take $\mathcal{L} = \{ww : w \in \{0,1\}^*\}$.
- ▶ \mathcal{L} is **not** a CFL (why?)
- ▶ We prove that $\overline{\mathcal{L}}$ **is** a CFL

Complementation cont.

- ▶ For any $y \in \bar{\mathcal{L}}$, either
 - ▶ y 's length is **odd**, or
 - ▶ y 's length is **even**, 2ℓ , and $\exists i \geq 1$ such that $y_i \neq y_{\ell+i}$.
- ▶ It suffices to construct a PDA/CFG for $\bar{\mathcal{L}}_{\text{even}}$ – the **even length** members of $\bar{\mathcal{L}}$ (**why?**)
- ▶ Let $\bar{\mathcal{L}}_{\text{even}}^{\sigma} = \{ \{0, 1\}^k \sigma \{0, 1\}^j \{0, 1\}^k \bar{\sigma} \{0, 1\}^j : k, j \geq 0 \}$
- ▶ Note that $\bar{\mathcal{L}}_{\text{even}} = \bar{\mathcal{L}}_{\text{even}}^0 \cup \bar{\mathcal{L}}_{\text{even}}^1$
- ▶ and that $\bar{\mathcal{L}}_{\text{even}}^{\sigma} = \{ \{0, 1\}^k \sigma \{0, 1\}^k \{0, 1\}^j \bar{\sigma} \{0, 1\}^j : k, j \geq 0 \}$
- ▶ CFG for $\bar{\mathcal{L}}_{\text{even}}^0$
 - $S \rightarrow AB$
 - $A \rightarrow CAC \mid 0$
 - $B \rightarrow CBC \mid 1$
 - $C \rightarrow 0 \mid 1$

A PDA for $\overline{\mathcal{L}}_{\text{even}}^0$

Idea: Guess $k, j \geq 0$, and accept w if it is of the form:
 $\{0, 1\}^k 0 \{0, 1\}^k \{0, 1\}^j 1 \{0, 1\}^j$



A simpler PDA:

- ▶ Loops from q_1 and q_3 changed to $0/1, \epsilon \rightarrow 2$
- ▶ Loops from q_2 and q_4 changed to $0/1, 2 \rightarrow \epsilon$

Homomorphism and inverse homomorphism

- ▶ *Homomorphism*: replaces each **letter** with a **word**
- ▶ **Example**: $h(1) = aba$, $h(0) = aa$
 $h(010) = aa\ aba\ aa$
 $\mathcal{L}_1 = \{0^n 1^n \mid n \geq 0\}$, $h(\mathcal{L}_1) = \{a^{2n}(aba)^n \mid n \geq 0\}$.
- ▶ Claim: Assuming that \mathcal{L} is a CFL, then so is $h(\mathcal{L})$
- ▶ Proof? use the grammar
- ▶ *Inverse homomorphism*: $h^{-1}(w) = \{x : h(x) = w\}$,
 $h^{-1}(\mathcal{L}) = \{x : h(x) \in \mathcal{L}\}$
- ▶ **Example**: $\mathcal{L}_2 = \{a^n b^n a^i \mid n, i \geq 0\}$, $h^{-1}(\mathcal{L}_2) = \{10^i, 0^i \mid i \geq 0\}$.
- ▶ Claim: Assuming that \mathcal{L} is a CFL, then so is $h^{-1}(\mathcal{L})$
- ▶ Proof? use the automata (see next)

CFL's are closed under inverse homomorphism

- ▶ Idea: let P be a PDA for \mathcal{L} . On input word w , emulate $P(h(w))$
- ▶ But we cannot afford to store $h(w)$!
- ▶ Solution, compute $h(w)$ "on demand":

Algorithm 9

Input w :

1. Initialize a "buffer" Buff to $h(a)$, where a is the first letter of w
 2. Emulate a running of P with Buff as its input string.
Each time Buff is fully read by P , set $\text{Buff} = h(a)$, where a is the next letter in w (if exists)
 3. Accept iff P does
- ▶ How do we implement Buff ?

CFL's are closed under inverse homomorphism, cont.

Given PDA $(Q, \Sigma, \Gamma, \delta, q_0, F)$ and homomorphism $h: \Delta \rightarrow \Sigma^*$, we define a new PDA $(Q', \Delta, \Gamma, \delta', q'_0, F')$.

- ▶ Let $k = \max_{a \in \Delta} |h(a)|$ and $\tilde{\Sigma} = \cup_{0 \leq i \leq k} \Sigma^i$.
- ▶ $Q' = Q \times \tilde{\Sigma}$.
- ▶ $q'_0 = [q_0, \varepsilon]$.
- ▶ $F' = F \times \{\varepsilon\}$.
- ▶ δ' is define as follows:
 - ▶ if $(p, \gamma) \in \delta(q, \varepsilon, \xi)$, then $([p, x], \gamma) \in \delta'([q, x], \varepsilon, \xi)$, for any $x \in \tilde{\Sigma}$.
 - ▶ if $(p, \gamma) \in \delta(q, a, \xi)$, then $([p, x], \gamma) \in \delta'([q, ax], \varepsilon, \xi)$, for any $x \in \tilde{\Sigma}$.
 - ▶ $([q, h(a)], \xi) \in \delta'([q, \varepsilon], a, \xi)$, for any $a \in \Delta$ and $\xi \in \Gamma$.

Part III

Algorithmic Questions

Emptiness of CFGs

Question 10

Given a CFG, G , is $\mathcal{L}(G) = \emptyset$?

In other words, is there a string generated by G ?

Theorem 11

There is an algorithm that solves this problem (and always halts).

Possible approaches for a proof:

- ▶ **Bad Idea:** We know how to test whether $w \in \mathcal{L}(G)$ for any string w , so just try it for each w ...
- ▶ **Better Idea:** Can the **start variable** generate a string of **terminals**?
- ▶ **A more holistic approach:** Can a particular variable generate a string of **terminals**?

Checking emptiness

Idea: Mark variables that can produce a string of terminals

Algorithm 12 (Deciding $\mathcal{L}(G) = \emptyset$)

1. Mark all terminal symbols in G .
2. Repeat until no new variable become marked:
Mark any A where $A \rightarrow U_1 U_2 \dots U_k$ and all U_i have already been marked.
3. Remove all **unmarked** variables, and any rule they appear in.
4. If S is removed, then $\mathcal{L}(G) = \emptyset$.

Correctness?

CFGs fullness

Question 13

Given a CFG G , is $\mathcal{L}(G) = \Sigma^*$?

- ▶ We just saw an algorithm to determine, given a CFG G , whether $\mathcal{L}(G) = \emptyset$
- ▶ $\mathcal{L}(G) = \Sigma^*$ iff $\overline{\mathcal{L}(G)} = \emptyset$. Why not modify the algorithm so it determines emptiness of the **complement**?
- ▶ Unfortunately, CFGs are not closed under complement.

Fact 14

There is **no** algorithm to solve **CFG fullness**.

- ▶ We are not prepared to prove this remarkable fact (**yet**).

Finiteness of CFGs

Question 15

Given a CFG G , is $|\mathcal{L}(G)|$ finite?

First, a useful subroutine.

Algorithm 16 (Removing redundant variables and terminals)

1. Mark all terminal symbols in G .
2. Repeat until no new variable become marked:
Mark any A where $A \rightarrow U_1 U_2 \dots U_k$ and all U_i have already been marked.
3. Remove all **unmarked** variables, and any rule they appear in.
4. If S is removed, then $\mathcal{L}(G) = \emptyset$.
5. Remove any variable A not reachable from S .
6. Remove any terminal which does not appear in some rule.

Back to finiteness of CFGs

Question 17

Given a CFG G , is $|\mathcal{L}(G)|$ finite?

Algorithm 18

1. Remove redundant variables and terminals.
2. Turn into a CNF.
3. Create a graph C :
 - ▶ Node for each variable $v \in V$.
 - ▶ **Directed** edges $(A, B), (A, C)$ for each rule $A \rightarrow BC \in R$.
4. Return **TRUE** iff C has a no **cycles**.

Correctness?

Less efficient algorithm “using” the pumping lemma:

- ▶ $|\mathcal{L}(G)| = \infty$ iff $\exists w \in \mathcal{L}(G)$ with $\ell \leq |w| \leq 2\ell$ (?) (ℓ is the pumping length)
- ▶ Hence, it suffices to check whether $\mathcal{L}(G)$ has a word on such length

Inherent ambiguity

Question 19

Given a CFG G , is $\mathcal{L}(G)$ inherently ambiguous?

(i.e., for **any** CFG generating $\mathcal{L}(G)$ exists $w \in \mathcal{L}$ with two different parse trees).

Fact 20

*There is **no** algorithm to solve CFG inherent ambiguity.*

We will **not** prove this fact, yet you want to know it to put things in context.

When are two CFGs equivalent?

Question 21

Given two CFG G_1 and G_2 , test if $L(G_1) = L(G_2)$.
Is there an algorithm to solve this problem?

Part IV

Equivalence Theorem

The CFG–PDA equivalence theorem

Theorem 22

$$\mathcal{L}_{\text{PDA}} = \mathcal{L}_{\text{CFG}}.$$

(i.e., A language is context free **if and only if** some pushdown automata accepts it.)

This time (unlike the regular expression vs. regular languages theorem), both the proof “**if**” part and of the “**only if**” part are non trivial.

Proof sketch follows.

CFL \implies PDA

Lemma 23

$\mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}}$: any CFL has a PDA that accepts it.

- ▶ Let \mathcal{L} be a CFL, and let $G = (V, \Sigma, R, S)$ be a CFG for \mathcal{L}
- ▶ We build a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, such that on input w it “figures out” if there is a derivation of w using G .

Question 24

How does P figure out which substitution to make?

Answer: It **guesses**.

Simplifying assumptions

1. In a **single** move, a PDA can push a **whole** word (from some fixed set) into the stack (first letter at the top)

Can we justify it?

2. When deriving a word from a CFL, we always substitute the **left most** variable

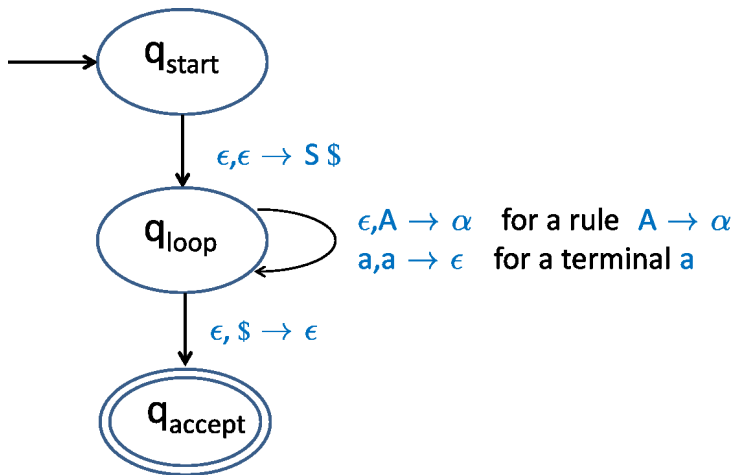
Does it change the derived language?

Informal description of P

Algorithm 25 (P)

1. Push $S\$$ on stack
2. While top of the stack t is not $\$$:
3. If t is variable A ,
(*non-deterministically*) select rule $A \rightarrow \alpha$ and substitute t with α .
4. If t is a terminal a ,
read next input and compare; **Reject** if different.
5. **Accept** if end of input and stack is empty.

State Diagram for P

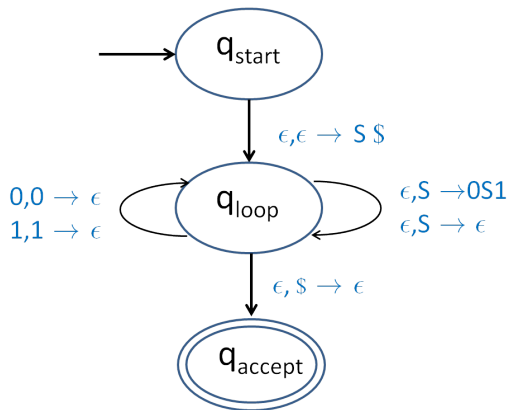


Example

consider the CFG:

$$S \rightarrow 0S1 \mid \epsilon.$$

The related PDA:



Claim: $\mathcal{L}(P) = \mathcal{L}(G)$

Claim 26

$S \xrightarrow{*} \alpha$ iff $\alpha = \alpha_1 \alpha_2$ such that $(q_{loop}, \alpha_2 \$) \in \widehat{\delta}(q_{loop}, \alpha_1, \$)$.

Does the above yields that $\mathcal{L}(P) = \mathcal{L}(G)$?

Note that $\alpha_1 \in \Sigma^*$

$S \xrightarrow{*} \alpha \implies \alpha = \alpha_1 \alpha_2$ **such that** $(q_{loop}, \alpha_2 \$) \in \widehat{\delta}(q_{loop}, \alpha_1, S \$)$

Proof by induction on the number of **derivations** steps used to yield α from S .

- ▶ Single derivation step: hence there is a rule $S \rightarrow \alpha$. Thus $(q_{loop}, \alpha \$) \in \widehat{\delta}(q_{loop}, \varepsilon, S \$)$, and the proof follows for $\alpha_1 = \varepsilon$ and $\alpha_2 = \alpha$.
- ▶ Assume $S \xrightarrow{*} \alpha$ in $k > 1$ derivation steps, and let α' be the string derived by the first $(k - 1)$ steps.
- ▶ By i.h $\alpha' = \alpha'_1 \alpha'_2$ such that $(q_{loop}, \alpha'_2 \$) \in \widehat{\delta}(q_{loop}, \alpha'_1, S \$)$
- ▶ Write $\alpha'_2 = w_1 A w_2$ where A is the **left most variable** in α'_2 .
- ▶ k 'th derivation step replaces this occurrence of A with a string s (?)
- ▶ It is easy to see that $(q_{loop}, s w_2 \$) \in \widehat{\delta}(q_{loop}, \alpha'_1 w_1, S \$)$.
- ▶ To complete the proof take $\alpha_1 = \alpha'_1 w_1$ and $\alpha_2 = s w_2$.

$\alpha = \alpha_1\alpha_2$ **such that** $(q_{loop}, \alpha_2\$) \in \widehat{\delta}(q_{loop}, \alpha_1, \$) \implies S \xrightarrow{*} \alpha$

Proof by induction on the number of **steps** used by P to process α_1 .

- ▶ Single step: $\alpha_1 = \varepsilon$ and $\alpha_2 = S\$$, and the proof follows since $S \xrightarrow{*} S$.
- ▶ Assume α_1 was processed in $k > 1$ steps. Let α'_1 and α'_2 be **input string read** and **stack value before last step**:
i.e., By definition, $(q_{loop}, \alpha_2\$) \in \widehat{\delta}(q_{loop}, a, \alpha'_2\$)$, for $a = \varepsilon$ if $\alpha_1 = \alpha'_1$, and last letter of α_1 o/w.
- ▶ $(q_{loop}, \alpha'_2\$) \in \widehat{\delta}(q_{loop}, \alpha'_1, \$)$.
- ▶ By i.h $S \xrightarrow{*} \alpha' = \alpha'_1\alpha'_2$.
- ▶ If k 'th move of P is **reading an input character**, then $\alpha_1\alpha_2 = \alpha'_1\alpha'_2$, and therefore $S \xrightarrow{*} \alpha_1\alpha_2$
- ▶ O/w, $\alpha'_1 = \alpha_1$, $\alpha'_2 = Aw$ and $\alpha_2 = sw$ for some rule $A \rightarrow s \in R$
- ▶ Hence $S \xrightarrow{*} \alpha_1\alpha_2$

Lemma 27

$$\mathcal{L}_{\text{PDA}} \subseteq \mathcal{L}_{\text{CFG}}.$$

If a PDA accepts a language then it is context free.

We prove the lemma by constructing a CFG G for a language \mathcal{L} accepted by a PDA P

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$. We assume wlg. that:

- ▶ A **single** accepting state $q_a \in F$.
- ▶ P **empties** the stack before accepting
- ▶ Each transition **either pops or pushes**

Can we justify the above?

Proof idea

- ▶ Suppose string x takes P from state p with empty stack to state q with empty stack.
- ▶ First move that touches the stack must be a push, last must be a pop.
- ▶ *Either stack is empty only at start and finish:*

Simulate by $A_{pq} \rightarrow aA_{rs}b$, where a, b are first and last symbols in x , r is state that p can reach in a step and s is state that can reach q in a step.

- ▶ *Or stack was empty at some point in between:*

Simulate by $A_{pq} \rightarrow A_{pr}A_{rq}$ where r is intermediate state and P has empty stack.

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

▶ $V = \{A_{pq} : p, q \in Q\}$

Idea: A_{pq} will generate **all strings** that take P from p with an **empty stack**, to q with an **empty stack**

▶ $S = A_{q_0, q_a}$

▶ Initially $R = \emptyset$ and

1. Add $\{A_{pq} \rightarrow A_{p,r}A_{r,q} : p, q, r \in Q\}$ to R

2. Add $\{A_{qq} \rightarrow \varepsilon : q \in Q\}$ to R

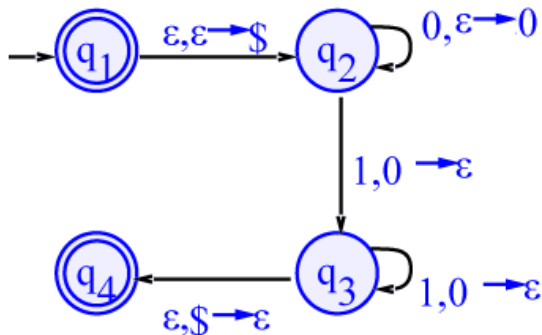
3. For all $p, r, s, q \in Q$, $a, b \in \Sigma_\varepsilon$ and $\gamma \in \Gamma$ such that

3.1 $(r, \gamma) \in \delta(p, a, \varepsilon)$ and

3.2 $(q, \varepsilon) \in \delta(s, b, \gamma)$

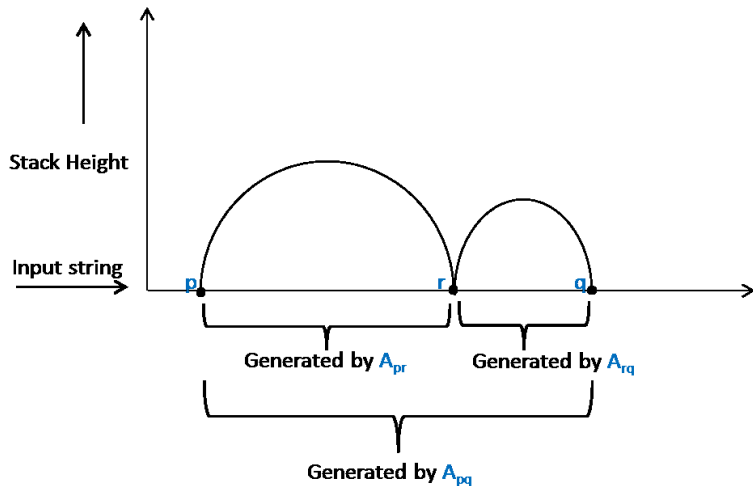
add $A_{pq} \rightarrow aA_{r,s}b$ to R

Example PDA to CFG

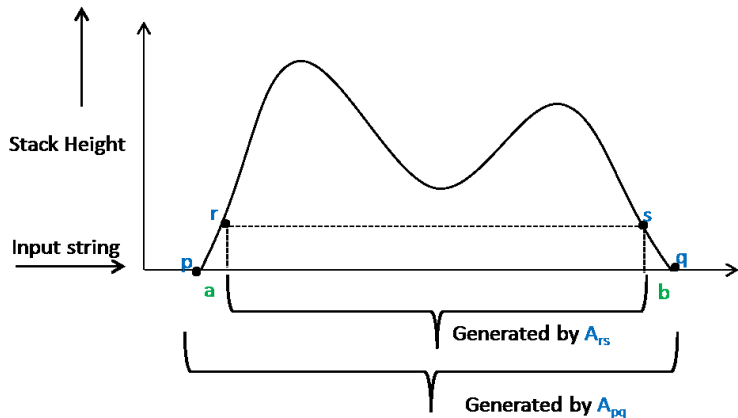


$$\begin{aligned} A_{q_1, q_4} &\rightarrow A_{q_2, q_3} \\ A_{q_2, q_3} &\rightarrow 0A_{q_2, q_3} 1 \\ A_{q_2, q_3} &\rightarrow 0A_{q_2, q_2} 1. \\ A_{q_2, q_2} &\rightarrow \varepsilon. \end{aligned}$$

PDA Computation corresponding to $A_{pq} \rightarrow A_{p,r}A_{r,q}$



PDA Computation corresponding to $A_{pq} \rightarrow aA_{r,s}b$



Claim: $\mathcal{L}(G) = \mathcal{L}(P)$

Claim 28

$A_{pq} \xrightarrow{*} w \in \Sigma^*$ iff $(q, \varepsilon) \in \widehat{\delta}(p, w, \varepsilon)$

Proof by induction on the number of derivation rules/ transitions

A short summary

- ▶ Regular Languages \equiv Finite Automata.
- ▶ Context Free Languages \equiv Push Down Automata.
- ▶ Closure properties of regular languages and of CFLs.
- ▶ Most algorithmic problems for finite automata are solvable.
- ▶ Some algorithmic problems for finite automata are not solvable.
- ▶ Pumping lemmata for both classes of languages.
- ▶ There are additional languages out there.

View over the horizon

