Computational Models — Lecture 5[1]

Handout Mode

Ronitt Rubinfeld and Iftach Haitner.

Tel Aviv University.

March 28/30, 2016

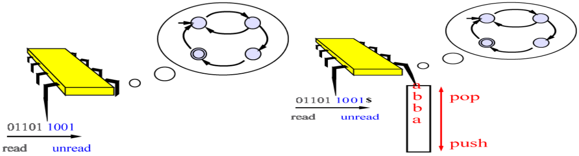
[1] Based on frames by Benny Chor, Tel Aviv University, modifying frames by Maurice Herlihy, Brown University. Also including modifications of Yishay Mansour.

Talk Outline

- Algorithmic issues for CFL
- Chomsky Normal Form
- Pumping Lemma for context free languages
- Push Down Automata (PDA)

Next week:

- Equivalence of CFGs and PDAs



- Sipser's book, 2.1, 2.2 & 2.3

Last time

- context-free languages
- context-free grammars

Formal Definition

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

- V is a finite set of variables
- Σ is a finite set of terminals
- R is a finite set of rules: each rule is a variable and a finite string of variables and terminals.
- S is the start symbol.
- If u and v are strings of variables and terminals, and $A \to w$ is a rule of the grammar, then uAv yields uwv, written $uAv \to uwv$.
- $u \xrightarrow{*} v$ if $u = v$, or $u \to u_1 \to \ldots \to u_k \to v$ for some sequence $u_1, u_2, \ldots, u_k$

Definition 1

The language of the grammar G, denoted $\mathcal{L}(G)$, is $\{w \in \Sigma^* : S \xrightarrow{*} w\}$

where $\xrightarrow{*}$ is determined by G.

Part I

Checking Membership

Checking Membership in a CFL

Challenge

Given a CFG G and a string w, decide whether $w \in \mathcal{L}(G)$?

Initial Idea: Design an algorithm that tries all derivations.

Problem: If G does not generate w, we'll never stop.

Possible solution: Use special grammars that are:

- just as expressive!
- better for checking membership.

Chomsky Normal Form (CNF)

A simplified, canonical form of context free grammars.
$G = (V, \Sigma, R, S)$ is in a CNF, if every rule in in R has one of the following forms:

$$
\begin{array}{ll}
A \to a, & A \in V \;\; \wedge \;\; a \in \Sigma \\
A \to BC, & A \in V \;\; \wedge \;\; B, C \in V \setminus \{S\} \\
S \to \varepsilon. &
\end{array}
$$

Simpler to analyze: each derivation adds (at most) a single terminal, S only appears once, ε appears only at the empty word

What does parse tree look like?

Most internal nodes are degree 2 (except parents of leaves, which are degree 1)

CNF: Theorem

Theorem 2

Any context-free language is generated by a context-free grammar in Chomsky Normal Form.

Proof Idea:

- Add new start symbol S_0.
- Convert “long rules” to proper form.
- Eliminate all ε rules of the form $A \to \varepsilon$.
- Eliminate all “unit” rules of the form $A \to B$.
- Patch up rules so that grammar generates the same language.

Add new start symbol

Add new start symbol S_0 and rule $S_0 \to S$

(Guarantees that new start symbol is never on right hand side of a rule)
e.g.

$$\begin{aligned} S &\to A \mid ab \mid \varepsilon \\ A &\to baA \mid S \end{aligned}$$

becomes

$$\begin{aligned} S_0 &\to S \\ S &\to A \mid ab \mid \varepsilon \\ A &\to baA \mid S \end{aligned}$$

Convert "long rules": Phase 1 – no mixing of terminals/nonterminals, and no multiple terminals

$$
\begin{aligned}
S &\rightarrow ccAbA \mid bc \mid b \\
A &\rightarrow a \mid bb
\end{aligned}
$$

becomes

$$
\begin{aligned}
S &\rightarrow X_c X_c A X_b A \mid X_b X_c \mid b \\
A &\rightarrow a \mid X_b X_b \\
X_c &\rightarrow c \\
X_b &\rightarrow b
\end{aligned}
$$

Convert "long rules": Phase 2 – multiple nonterminals

$$S \rightarrow AAAB$$

becomes

$$\begin{aligned} S &\rightarrow AN_1 \\ N_1 &\rightarrow AN_2 \\ N_2 &\rightarrow AB \end{aligned}$$

Eliminate "ε-rules"

Repeat until all $A \to \epsilon$ rules are gone:

- remove $A \to \varepsilon$
- for any rule of form $R \to AB$ or $R \to BA$, add $R \to B$.
- for any rule of form $R \to AA$ add $R \to A$ and $R \to \varepsilon$ (unless $R \to \varepsilon$ has already been removed).
- for any rule of form $R \to A$ add $R \to \varepsilon$ (unless $R \to \varepsilon$ has already been removed.)

(Alternative description: Let W be the set of variables A such that $A \xrightarrow{*} \epsilon$. For each $A \in W$, (1) remove $A \to \varepsilon$ if present, (2) for any rule of form $R \to AB$ or $R \to BA$, add $R \to B$, even if $A = B$. Don't need to add $R \to \varepsilon$ since $R \in W$).

Eliminate "unit rules"

Repeat until all unit rules removed

- remove some $A \to B$
- for each $B \to U$ add $A \to U$ (unless $A \to U$ was previously removed unit rule)

(Alternative description: Create a directed graph with nodes corresponding to variables and edge from A to B if $A \to B$ is a rule. For each strong component in the graph, replace all variables by a single variable.)

Cleanup

Delete all “unreachable” rules, e.g.:

- delete all $A \to A$ rules
- for each rule with A on LHS, make sure that A appears on RHS of some rule that is reachable from start variable.
- for each rule with A on RHS, make sure that A also appears on LHS of a rule.
- for each variable A, make sure it can reach a terminal.

CNF: Example

$$\begin{aligned} S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

Is transformed into:

$$\begin{aligned} S_0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\ S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\ A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\ A_1 &\rightarrow SA \\ U &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

CNF: Bounded Derivation Length

Lemma 3

For a CNF grammar G and $w \in \mathcal{L}(G)$ with $|w| = n \geq 1$, it holds that w has a derivation of length $2n - 1$.

Proof? consider the parsing tree for w

Advantage: Easier to check whether $w \in \mathcal{L}(G)$

Checking Membership for Grammars in CNF Form

Given a CNF grammar $G = (V, \Sigma, R, S)$, we build a function $\text{Derive}(A, x)$ that returns TRUE iff $A \xrightarrow{*} x$.

Algorithm 4 ($\text{Derive}(A, x)$)

- If $x = \varepsilon$: if $A \to \varepsilon \in R$ (i.e., $A = S$) return TRUE, otherwise return FALSE.
- If $|x| = 1$: if $A \to x \in R$ return TRUE, otherwise return FALSE.
- For each $A \to BC$ and each partition $x = x_L x_R$ (i.e. $x_L = x_1 \ldots x_j$ and $x_R = x_{j+1} \ldots x_{|x|}$):
  - Call $\text{Derive}(B, x_L)$ and $\text{Derive}(C, x_R)$.
  - Return TRUE if *both* return TRUE.
- Return FALSE.

Test whether $w \in \mathcal{L}(G)$ by calling $\text{Derive}(S, w)$

Correctness?

- Procedure Derive can also output a parse tree for w
- Have we critically used that G is in CNF?

Time Complexity of Derive

What is the time complexity $T\colon \mathbb{N} \mapsto \mathbb{N}$ of Derive?

- Each recursive call tests $|R|$ rules and n partitions.
- $T(n) \leq |R| \cdot n \cdot 2T(n-1)$
- $T(n) \in O((|R| \cdot n)^n)$.

Still exponential...

Efficient Algorithm

- Keep in memory the results of Derive(A, x).
  - Main observation: Number of different inputs is $|V| \cdot n^2$. why???
  - Only $|V| \cdot n^2$ calls, each takes $O(|R| \cdot n)$.
  - $T(n) \in O(|R| \cdot n^3 \cdot |V|)$.
- Polynomial time!
- This approach is called Dynamic Programming

  Basic idea:
  - If number of different inputs is limited, say $I(n)$.
  - Each run (excluding recursive calls) takes at most $R(n)$ time
  - Total running time is bounded by $T(n) \leq R(n)I(n)$.

Part II

Non-Context-Free Languages

Proving a Language is not a CFL

- The pumping lemma for finite automata and Myhill-Nerode theorem are our tools for showing that languages are not regular.
- We will now show a similar pumping lemma for context-free languages.
- It is slightly more complicated ...

Pumping Lemma for CFL (also known as, the *uvxyz* Theorem)

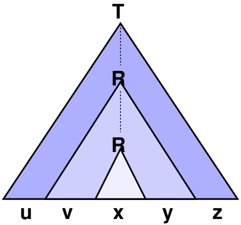
Theorem 5

For any CFL $\mathcal{L}$ *there exists* $\ell \in \mathbb{N}$ *("critical length"), such that for any* $w \in \mathcal{L}$ *with* $|w| \geq \ell$*, there exist* $u, v, x, y, z \in \Sigma^*$ *such that* $w = uvxyz$ *and*

- *For every* $i \geq 0$*:* $uv^ixy^iz \in \mathcal{L}$
- $|vy| > 0$*, ("non-triviality")*
- $|vxy| \leq \ell$ *(extra property that is helpful for us later!)*

Basic Intuition

Let $\mathcal{L}$ be a CFL and a let w be a "very long" string in $\mathcal{L}$. Then w must have a "tall" parse tree.



Hence, some root-to-leaf path must repeat a symbol. Why is that so?
We have: $T \xrightarrow{*} uRz$, $R \xrightarrow{*} vRy$, and $R \xrightarrow{*} x$.
But then the second R could also produce vRy, giving uv^2xy^2z!

Proof of Thm 5

Let G be a CFG and let $\mathcal{L} = \mathcal{L}(G)$.

- Let b be the max number of symbols in right-hand-side of any rule (what is b for a CNF grammar?).

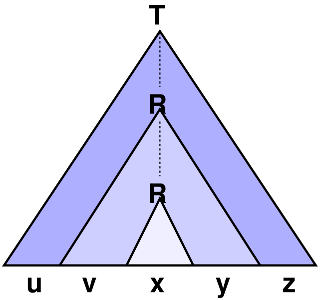
  Since no node in a parse tree of G has more than b children, at depth d such tree has at most b^d leaves.
- Let $|V|$ be the number of variables in G, and set $\ell = b^{|V|+2}$.

Let w be a string with $|w| \geq \ell$, and let T be parse tree for w (with respect to G) with fewest nodes

- T has height $\geq |V| + 2$
- Some path in T has length $\geq |V| + 2$
- Such path repeats a variable R

Proof of Thm 5 cont.

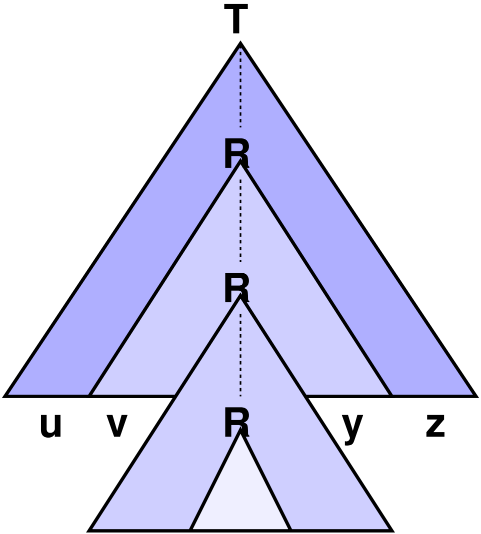
Set $w = uvxyz$



- Each occurrence of R produces a string
- Upper produces string vxy
- Lower produces string x

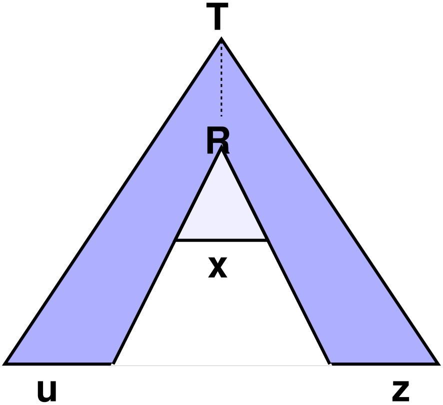
Proving $uv^ixy^iz \in \mathcal{L}$ for all $i > 1$

Replacing smaller by larger yields uv^ixy^iz, for $i > 0$.



Proving $uv^ixy^iz \in \mathcal{L}$ for $i = 0$

Replacing larger by smaller yields uxz.

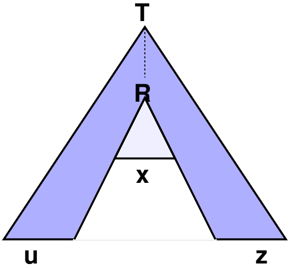


Together, they establish:

- $uv^ixy^iz \in \mathcal{L}$ for all $i \geq 0$

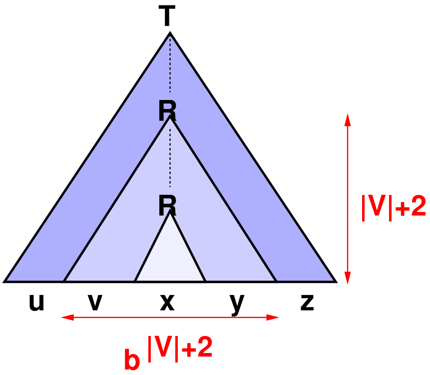
Proving $|vy| > 0$

If v and y are both ε, then



is a parse tree for w with fewer nodes than T, a contradiction.

Proving $|vxy| \le \ell$



- Without loss of generality both occurrences of R lie in bottom $|V|+1$ variables on the path.
- The upper occurrence of R (from now on R^1) generates vxy.
- Subtree rooted at R^1 is of height at most $|V|+2$.

  Hence, $|vxy| \le b^{|V|+2} = \ell$.

Non CFL Example (1)

Claim 6

$\mathcal{L}_1 = \{a^n b^n c^n : n \in \mathbb{N}\}$ is not a CFL.

Proof: By contradiction. Assume $\mathcal{L}_1$ is a CFL with grammar G, let ℓ be the critical length of G and consider $w = a^\ell b^\ell c^\ell$. Let u, v, x, y, z be the strings with $w = uvxyz$ guaranteed by Thm 5 for w.

- Note that neither v nor y contain
  - both a's and b's, or
  - both b's and c's,

  (otherwise uv^2xy^2z would have out-of-order symbols).
- But if v and y contain only one letter, then uv^2xy^2z is imbalanced

♣

Non CFL Example (2)

Claim 7

$\mathcal{L}_2 = \{a^i b^j c^k : 0 \leq i \leq j \leq k\}$ is not context free.

Proof: By contradiction. Assume $\mathcal{L}_2$ is a CFL with grammar G, let ℓ be the critical length of G and consider $w = a^\ell b^\ell c^\ell$. Let u, v, x, y, z be the strings with $w = uvxyz$ guaranteed by Thm 5 for w.

- Neither v nor y contains two distinct symbols, because otherwise uv^2xy^2z would have out-of-order symbols.
- vxy cannot be all the same letter (if a or b, can pump "up", if c can pump "down").
- $|vxy| \leq \ell$, so either
  - v contains only a's and y contains only b's, but then uv^2xy^2z has too few c's.
  - v contains only b's and y contains only c's, but then uv^0xy^0z has too many a's.

♣

Non CFL Example (3)

Claim 8

$\mathcal{L}_3 = \{ww \colon w \in \{0,1\}^*\}$ is not context-free.

Proof:
By contradiction. Assume $\mathcal{L}_3$ is a CFL with grammar G, let ℓ be the critical length of G and consider $w = 0^\ell 1^\ell 0^\ell 1^\ell$. Let u, v, x, y, z be the strings with $w = uvxyz$ guaranteed by Thm 5 for w.

- Recall that $|vxy| \leq \ell$
- Assuming vxy is in the first half of w, then uv^2xy^2z “moves" a 1 into the first position of second half.
- Assuming vxy is in the second half, then uv^2xy^2z “moves" a 0 into the last position of first half.
- Assuming vxy straddles the midpoint, then pumping *down* to uxz yields $0^\ell 1^i 0^j 1^\ell$ where i and j cannot both be ℓ.

♣

Note that $\{ww^R \colon w \in \{0,1\}^*\}$ is a CFL.

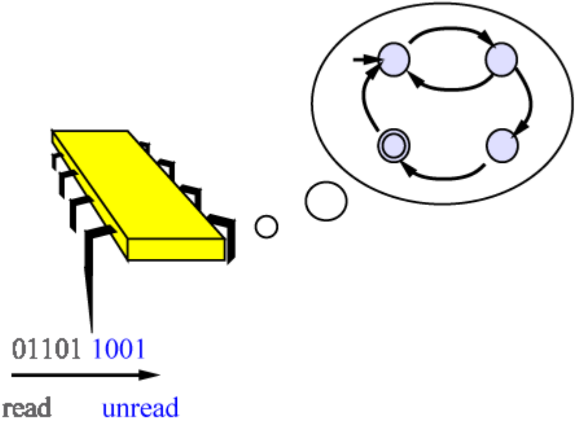
Part III

Push-Down Automata

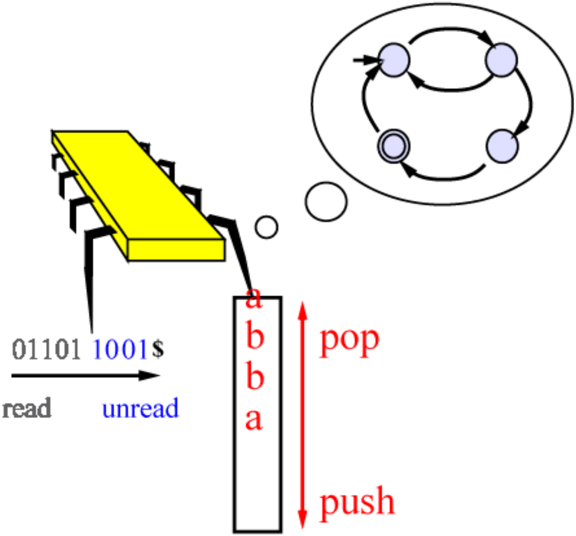
String Generators and String Acceptors

- 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*

A Finite Automaton



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

Informally:

- Read and push a's
- Either pop and match with b's or pop and match with c's

A non-deterministic choice

Pushdown Automaton (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,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is the transition function,[2]
- $q_0 \in Q$ is the starting state, and
- $F \subseteq Q$ is the set of accepting states.

[2] $X_\varepsilon := X \cup \{\varepsilon\}$.

The language accepted by a PDA

- A pushdown automaton (PDA) M accepts a string w, if there is a "computation" of M on w (see next slide) that leads to an accepting state.
- The language accepted by M, denoted $\mathcal{L}(M)$, is the set of all strings $w \in \Sigma^*$ accepted by M.
- A (non-deterministic) PDA may have many computations on a single string

Model of Computation

The following is with respect to $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$.

Definition 9 (δ^*)

For $w \in \Sigma^*$ let $\widehat{\delta}(q, w, s)$ be all pairs $(q', s') \in Q \times \Gamma^*$ for which exist $w'_1, \ldots, w'_m \in \Sigma_\varepsilon$, states $r_1, \ldots, r_m \in Q$ and strings $s_0, s_1, \ldots s_m \in \Gamma^*$ s.t.:

1. $w = w'_1, \ldots, w'_m$, $r_0 = q$, $r_m = q'$, $s_0 = s$ and $s_m = s'$
2. For every $i \in \{0, \ldots, m-1\}$ exist $a, b \in \Gamma_\varepsilon$ and $t \in \Gamma^*$ s.t.:
   - 2.1 $(r_{i+1}, b) \in \delta(r_i, w'_{i+1}, a)$
   - 2.2 $s_i = at$ and $s_{i+1} = bt$

Namely, $(q', s') \in \widehat{\delta}(q_0, w, \varepsilon)$ if after reading w (possibly with in-between ε moves), M can find itself in state q' and stack value s'.

- M accepts $w \in \Sigma^*$ if $\exists q' \in \mathcal{F}$ such that $(q', t) \in \widehat{\delta}(q_0, w, \varepsilon)$ for some t.

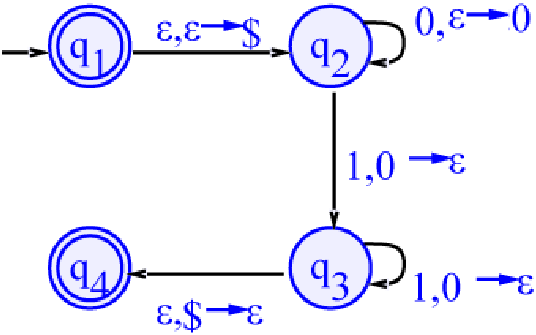
Diagram Notation

When drawing the automata diagram, we use the following notation

- Transition $a, b \to c$ 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 $\mathcal{L}_1 = \{0^n 1^n \colon n \geq 0\}$



Claim 10

$0011 \in L(P)$.

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 $\mathcal{L}_1 = \{0^n1^n : n \geq 0\}$

We want to show that $L(P) = \mathcal{L}_1 = \{0^n1^n : n \geq 0\}$
What do we need to prove?

Claim 11

- $\widehat{\delta}(q_1, \varepsilon, \varepsilon) = \{(q_1, \varepsilon), (q_2, \$)\}$.
- $\widehat{\delta}(q_1, 0^k, \varepsilon) = \{(q_2, 0^k\$)\}$, for $k \geq 1$.
- $\widehat{\delta}(q_1, 0^k1^i, \varepsilon) = \{(q_3, 0^{k-i}\$)\}$, for $k > i \geq 1$.
- $\widehat{\delta}(q_1, 0^k1^k, \varepsilon) = \{(q_3, \$), (q_4, \varepsilon)\}$, for $k \geq 1$.
- $\widehat{\delta}(q_1, w, \varepsilon) = \emptyset$, for $w \notin \{0^k1^i | k \geq i \geq 0\}$.

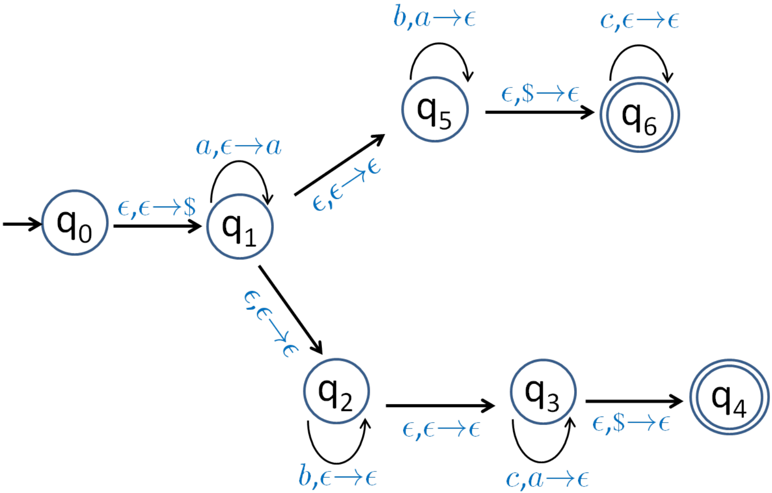
Knowing when stack is empty

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 deicide 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: $\mathcal{L} = \{x^n y^n : n \geq 0\} \cup \{x^n y^{2n} : n \geq 0\}$ is accepted by a non-deterministic PDA, but not by a deterministic one. (Proof? Book!)

PDA Languages

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

- $\mathcal{L}_{\text{PDA}} = \{\mathcal{L}(M) \colon M \text{ is a PDA}\}$

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

- $\mathcal{L}_{\text{CFG}} \subseteq \mathcal{L}_{\text{PDA}}$?
- $\mathcal{L}_{\text{PDA}} \subseteq \mathcal{L}_{\text{CFG}}$?
- $\mathcal{L}_{\text{PDA}} = \mathcal{L}_{\text{CFG}}$!!!