Harvard CS 121 and CSCI E-121

Lecture 9: Ambiguity, Pushdown Automata

Harry Lewis

October 1, 2013

- **Reading:** Sipser, §2.2.

A grammar for $L = \{x \in \{a, b\}^* : x$ has the same # of a's and b's$\}$

- $G = (\{S\}, \{a, b\}, R, S)$ where R has rules:

$$S \to \varepsilon$$

$$S \to SS$$

$$S \to aSb$$

$$S \to bSa$$

Proof that $L(G) = \{x : x$ has the same # of a's and b's$\}$

(1) $x \in L(G) \Rightarrow x$ has the same # of a's and b's

Pf: Easy, every RHS has the same number of a's and b's.
Formal proof by induction on length k of the derivation.

(2) x has the same # of a's and b's $\Rightarrow x \in L(G)$

Proof: by induction on $|x|$

(a) $|x| = 0$

(b) $|x| = k + 2$

4 subcases depending on first and last symbols of x

(i) $x = ayb$ for some $y \in \Sigma^*$

(ii) $x = bya$ for some $y \in \Sigma^*$

(iii) $x = aya$ for some $y \in \Sigma^*$

(iv) Similar if $x = byb$ for some $y \in \Sigma^*$.

The $x = aya$ case

- Then $x = auva$, where u, v each has one more b than a, — that is, au and va each has the same number of as and bs
  - Because counting the excess of as over bs starting at the beginning of aya, the count starts at zero, increases by one at the first step, increases by one at the last step, ends at zero, and changes by ± 1 at each step.
- So by induction $S \Rightarrow^* au$, $S \Rightarrow^* va$, $S \Rightarrow SS \Rightarrow^* auva = x$

Balanced parentheses language

Balanced parentheses language

AKA “Dyck set over 2 letters”

$$R = \{S \to \varepsilon, S \to SS, S \to (S)\}$$

$$\begin{aligned} S \Rightarrow SS \Rightarrow (S)S \Rightarrow ((S))S \Rightarrow (())S \\ \Rightarrow (())(S) \Rightarrow (())() \end{aligned}$$

More examples of CFGs

- Arithmetic Expressions

G_1:

$$E \rightarrow x \mid y \mid E * E \mid E + E \mid (E)$$

G_2:

$$\begin{aligned} E &\rightarrow T \mid E + T \\ T &\rightarrow T * F \mid F \\ F &\rightarrow (E) \mid x \mid y \end{aligned}$$

Q: Which is “preferable”? Why?

Parse Trees

Derivations in a CFG can be represented by parse trees.

Examples:

Each parse tree corresponds to many derivations, but has unique leftmost derivation.

Parsing

Parsing: Given $x \in L(G)$, produce a parse tree for x. (Used to 'interpret' x. Compilers parse, rather than merely recognize, so they can assign semantics to expressions in the source language.)

Ambiguity: A grammar is ambiguous if some string has two parse trees.

Example:

Context-free Grammars and Automata

What is the fourth term in the analogy:

Regular Languages : Finite Automata

as

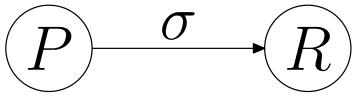
Context-free Languages : ???

Regular Grammars

Hint: There is a special kind of CFGs, the **regular grammars**, that generate exactly the regular languages.

A CFG is **(right-)regular** if any occurrence of a nonterminal on the RHS of a rule is as the rightmost symbol.

Turning a DFA into an equivalent Regular Grammar

- Variables are states.
- Transition $\delta(P, \sigma) = R$ 

  becomes $P \to \sigma R$
- If P is accepting, add rule $P \to \varepsilon$

Regular Grammars (cont.)

Example of DFA$\Rightarrow$ Regular Grammars:

$\{x : x$ has an even # of a's and an even # of b's$\}$.

Other Direction: Omitted.

Context-free Grammars and Automata

What is the fourth term in the analogy:

Regular Languages : Finite Automata

as

Context-free Languages : ???

Sheila Greibach, AB Radcliffe '60 *summa cum laude*

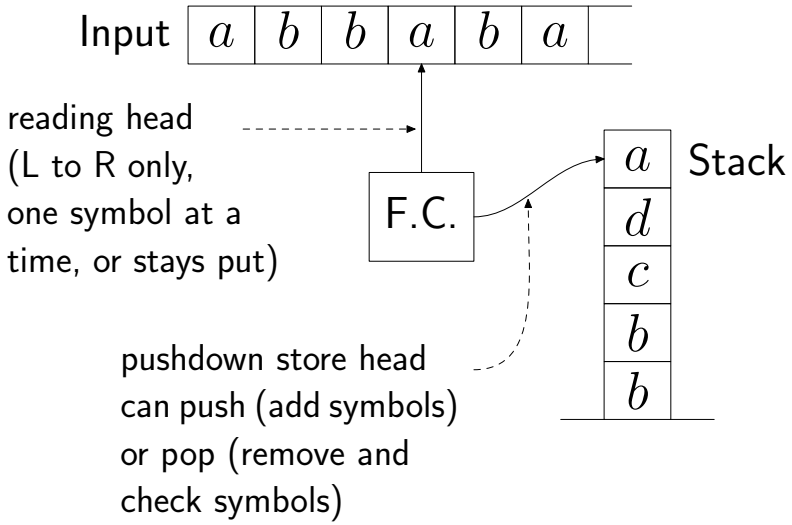
Inverses of Phrase Structure Generators

Harvard PhD Thesis, 1963

Pushdown Automata

= Finite automaton + "pushdown store"

- The pushdown store is a stack of symbols of unlimited size which the machine can read and alter only at the top.



Transitions of PDA are of form $(q, \sigma, \gamma) \mapsto (q', \gamma')$, which means:

If in state q with σ on the input tape and γ on top of the stack, replace γ by γ' on the stack and enter state q' while advancing the reading head over σ.

(Nondeterministic) PDA for "even palindromes"

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

$(q, a, \varepsilon) \mapsto (q, a)$	Push a's
$(q, b, \varepsilon) \mapsto (q, b)$	and b's
$(q, \varepsilon, \varepsilon) \mapsto (r, \varepsilon)$	switch to other state
$(r, a, a) \mapsto (r, \varepsilon)$	pop a's matching input
$(r, b, b) \mapsto (r, \varepsilon)$	pop b's matching input

So the precondition (q, σ, γ) means that

- the next $|\sigma|$ symbols (0 or 1) of the input are σ and
- the top $|\gamma|$ symbols (0 or 1) on the stack are γ

(Nondeterministic) PDA for "even palindromes"

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

$(q, a, \varepsilon) \mapsto (q, a)$	Push a's
$(q, b, \varepsilon) \mapsto (q, b)$	and b's
$(q, \varepsilon, \varepsilon) \mapsto (r, \varepsilon)$	switch to other state
$(r, a, a) \mapsto (r, \varepsilon)$	pop a's matching input
$(r, b, b) \mapsto (r, \varepsilon)$	pop b's matching input

Need to test whether stack empty: push $\$$ at beginning and check at end.

$(q_0, \varepsilon, \varepsilon) \mapsto (q, \$)$
$(r, \varepsilon, \$) \mapsto (q_f, \varepsilon)$

Language recognition with PDAs

A PDA accepts an input string

If there is a computation that starts

- in the start state
- with reading head at the beginning of string
- and the stack is empty

and ends

- in a final state
- with all the input consumed

A PDA computation becomes "blocked" (i.e. "dies") if

- no transition matches *both* the input and stack

Formal Definition of a PDA

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

  $Q =$ states

  $\Sigma =$ input alphabet

  $\Gamma =$ stack alphabet

  $\delta =$ transition function

  $$Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow P(Q \times (\Gamma \cup \{\varepsilon\})).$$

  $q_0 =$ start state

  $F =$ final states

Computation by a PDA

- M accepts w if we can write $w = w_1 \cdots w_m$, where each $w_i \in \Sigma \cup \{\varepsilon\}$, and there is a sequence of states $r_0, \ldots, r_m$ and stack strings $s_0, \ldots, s_m \in \Gamma^*$ that satisfy
  1. $r_0 = q_0$ and $s_0 = \varepsilon$.
  2. For each i, $(r_{i+1}, \gamma') \in \delta(r_i, w_{i+1}, \gamma)$ where $s_i = \gamma t$ and $s_{i+1} = \gamma' t$ for some $\gamma, \gamma' \in \Gamma \cup \{\varepsilon\}$ and $t \in \Gamma^*$.
  3. $r_m \in F$.
- $L(M) = \{w \in \Sigma^* : M \text{ accepts } w\}$.

PDA for $\{w \in \{a, b\}^* : \#_a(w) = \#_b(w)\}$

PDA for $\{w \in \{a,b\}^* : \#_a(w) = \#_b(w)\}$

Strategy:

- Keep $|\#_a(w) - \#_b(w)| = n$ on stack in form of $1^n\$$.
- Keep the sign of $\#_a(w) - \#_b(w)$ in the state:

  $+$ or $0 \Rightarrow$ state q_+

  $-$ or $0 \Rightarrow$ state q_-

The code

(s, e, e)	$\rightarrow$	$(p, \perp)$	Initialize
(p, a, e)	$\rightarrow$	$(p, 1)$	Keep count in $+$ state
$(p, b, 1)$	$\rightarrow$	(p, e)	”
$(p, b, \perp)$	$\rightarrow$	$(m, 1\perp)$	Switch to $-$ state
(m, b, e)	$\rightarrow$	$(m, 1)$	Keep count in $-$ state
$(m, a, 1)$	$\rightarrow$	(m, e)	”
$(m, a, \perp)$	$\rightarrow$	$(p, 1\perp)$	Switch to $+$ state
$(p, e, \perp)$	$\rightarrow$	(f, e)	Switch to final state
$(m, e, \perp)$	$\rightarrow$	(f, e)	”

$\rightarrow$ Nondeterministically switches to final state if stack is empty.

There are other schemes for this problem.