Lecture 10: Context-Free Grammars & Push-Down Automata

CSCI 101
Spring, 2019

Kim Bruce

Definitions

- A context-free grammar is a quadruple, G = (V, Σ, R, S) in which
  - V is a finite set of variables, containing terminals and nonterminals.
  - $\Sigma \subseteq V$ is the set of terminals
  - R is a finite set of productions of the form $U \to \alpha$, where U is a single nonterminal and α is a (possibly empty) string of terminals and nonterminals.
    I.e., OK to write $U \to \varepsilon$
  - S is an element of V called the start symbol.

Closure

- CFL's closed under
  - Concatenation
  - Kleene *
  - Reversal
  - Union
  - Substitution
- What about complement, intersection, difference, ...?

Derivations & Parse Trees

- A sequence of the form
  - $w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$ is called a derivation.
  - It is left-most if at each step, the left-most non-terminal is replaced using a rule of the grammar.
  - Similarly for right-most.
  - Does the distinction matter?
    - Not for meaning, but often important for parsers

Parse Trees

- A parse tree for grammar G = (V, Σ, R, S) is a rooted ordered tree in which
  - Every leaf node is labeled with an element of $\Sigma \cup \{\varepsilon\}$
  - The root node is labeled S
  - Every other node is labeled with an element of V - Σ
  - If m is a non-leaf node labeled X and the children of m are labeled $x_1, \ldots, x_n$ then R contains the rule $X \rightarrow x_1 \ldots x_n$

Example

Parse Trees & Derivations

- Given parse tree generally corresponds to several derivations of same string
  - E.g., left-most & right-most derivations
- Grammar G is ambiguous if there is at least one string in L(G) that has more than one parse tree. G is unambiguous otherwise.
  - I saw a man in the park with a telescope

- ... affects meaning
- Possible grammars for arithmetic expressions
  - Exp → Exp Op Exp | (Exp) | num
    Op → + | - | * | /

    versus

  - Exp → Exp Addop Term | Term
    Term → Term Mulop Factor | Factor
    Factor → (Exp) | num
    Addop → + | -
    Mulop → * | /

Pushdown Automata

Pushdown Automata

- A pushdown automaton is a sextuple, $(K, \Sigma, \Gamma, \Delta, s, A)$, where
  - K is a finite set of states,
  - Σ is a finite input alphabet,
  - Γ is a finite stack alphabet,
  - $s \in K$ is the start state, and
  - $\Delta \subseteq (K \times \Sigma \cup \{\varepsilon\} \times \Gamma^*) \times (K \times \Gamma^*)$,   *Non-deterministic!*
  - $A \subseteq K$ is the set of accepting, states.

Configurations

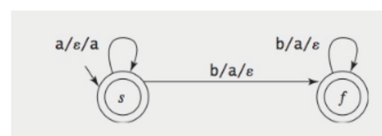
- A configuration of a pda M is an elt (q, w, γ) of $K \times \Sigma^* \times \Gamma^*$ representing the current state q, the input w left to be read, and the stack contents γ
  - Stack written from top down: c b a where c on top.
  - Initial configuration is (s,w,ε)
- Define $(q, cw, \gamma\, \gamma_{rest}) \vdash_M (q', w, \gamma'\, \gamma_{rest})$ iff $((q, c, \gamma), (q', \gamma')) \in \Delta$
- As usual $\vdash_M^*$ is reflexive, transitive closure

Accepting

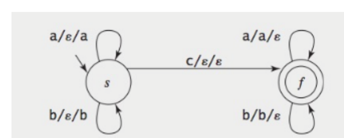
- A computation C of M is an accepting computation iff:
  - $C = (s, w, \varepsilon) \vdash_{M^*} (q, \varepsilon, \varepsilon)$, and
  - $q \in A$.   *Need empty stack and accepting state!*
- M accepts a string w iff at least one of its computations accepts.
  - $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$
- Note, in any configuration, can have 0, 1, 2, ... possible moves.

Example

- Pda for $\{a^n b^n \mid n \geq 0\}$



- Pda for $\{w\, c\, w^R \mid w\ \varepsilon\ \{a,b\}^*\}$



Deterministic

- A pda M is deterministic iff:
  - Δ_M contains no pairs of transitions that compete with each other, and
  - Whenever M is in an accepting configuration it has no available moves.
- Obvious pda for $\{w\, w^R \mid w \in \{a,b\}^*\}$ is not!

CFLs ≈ PDAs

- Theorem: The class of languages accepted by PDAs is exactly the class of context-free languages.

CFL ⇒ PDA

- Let G be a context-free grammar. Construct pda M s.t. $L(G) = L(M)$.
- Let $M = (\{p,q\}, \Sigma, V, \Delta, p, \{q\})$ where Δ contains:
  - $((p,\varepsilon,\varepsilon), (q,S))$
  - For each $X \to s_1 s_2 \ldots s_n$. in R, the transition: $((q, \varepsilon, X), (q, s_1 s_2 \ldots s_n))$. (*Type 1 rule*)
  - For each character $c \in \Sigma$, the transition: $((q, c, c), (q, \varepsilon))$. (*Type 2 rule*)
- Idea: Replace top-most non-terminal on stack by right side of production.

Building pda

- Example: $\{0^n 1^n \mid n \geq 0\}$
- Lemma: $(q,wx,S) \vdash^* (q,x,\gamma)$ iff $S \Rightarrow^* w\,\gamma$ in a left-most derivation.
  - $\Rightarrow$: Proof by induction on # steps in computation.
    - Base case trivial
    - Induction: Spose true for computations of length n, show for n+1
    - Two cases: last rule is type 1 or type 2
      Type 1: $(q,wx,S) \vdash^* (q,x,A\beta) \vdash (q,x,\alpha\beta)$ where $A \to \alpha$
      Type 2: $(q,yax,S) \vdash^* (q,ax,a\gamma) \vdash (q,x,\gamma)$

PDA accepts L(G)

- ⇐ Similar
- With lemmas, show L(G) = L(M) by taking x = ε
  - $(q,w,S) \vdash^* (q,\varepsilon,\varepsilon)$ iff $S \Rightarrow^* w$
  - and use opening transition $((p,\varepsilon,\varepsilon),(q,S))$
- Constructed pda is non-deterministic

PDA ⇒ CFG

- Harder
  - Requires converting pda into normal form pda
  - Non-terminal models parts of computation up to when input string removes item from stack
  - Important theoretically, but not in practice, so we'll skip it.

Algorithms for CFLs

Normal Forms

- Because of ε-productions, can be hard to determine if w in L.
  - Parsers recognize terms of language and build abstract syntax tree *(thinned down parse tree)*
- Normal forms can make it easier.
- Chomsky and Greibach Normal Forms
  - Do only Chomsky