CS 154 – Notes for Lectures 7 and 8 (4/17/00–4/24/00)

Ambiguous Grammars

A CFG is *ambiguous* if one or more terminal strings have multiple leftmost derivations from the start symbol.

- Equivalently: multiple rightmost derivations, or multiple parse trees.

Example

Consider $S \rightarrow AS \mid \epsilon$; $A \rightarrow A1 \mid 0A1 \mid 01$. The string 00111 has the following two leftmost deriviations from S:

1. $S \underset{lm}{\Rightarrow} AS \underset{lm}{\Rightarrow} 0A1S \underset{lm}{\Rightarrow} 0A11S \underset{lm}{\Rightarrow} 00111S \underset{lm}{\Rightarrow} 00111$
2. $S \underset{lm}{\Rightarrow} AS \underset{lm}{\Rightarrow} A1S \underset{lm}{\Rightarrow} 0A11S \underset{lm}{\Rightarrow} 00111S \underset{lm}{\Rightarrow} 00111$

- Intuitively, we can use $A \rightarrow A1$ first or second to generate the extra 1.

Inherently Ambiguous Languages

A CFL L is *inherently ambiguous* if *every* CFG for L is ambiguous.

- Such things exist; see course reader.

Example

The language of our example grammar is not inherently ambiguous, even though the grammar is ambiguous.

- Change the grammar to force the extra 1's to be generated last.

  $S \rightarrow AS \mid \epsilon$
  $A \rightarrow 0A1 \mid B$
  $B \rightarrow B1 \mid 01$

Why Care?

- Ambiguity of the grammar implies that at least some strings in its language have different structures (parse trees).
  - Thus, such a grammar is unlikely to be useful for a programming language, because two structures for the same string (program) implies two different meanings (executable equivalent programs) for this program.

- ❖ Common example: the easiest grammars for arithmetic expressions are ambiguous and need to be replaced by more complex, unambiguous grammars (see course reader).

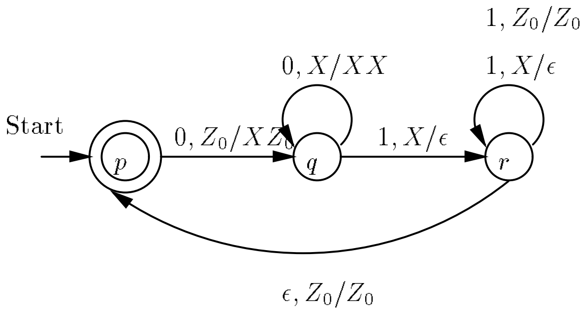
- An inherently ambiguous language would be absolutely unsuitable as a programming language, because we would not have any way of fixing a unique structure for all its programs.

Pushdown Automata

- Add a stack to a FA.
- Typically nondeterministic.
- An automaton equivalent to CFG's.

Example

Notation for "transition diagrams": $a, Z/X_1X +$ $2 \cdots X_k$ = "on input a, with Z on top of the stack, consume the a, make this state transition, and replace the Z on top of the stack by $X_1X_2 \cdots X_k$ (with X_1 at the top).



- p = starting to see a group of 0's and 1's; q = reading 0's and pushing X's onto the stack; r = reading 1's and popping X's until the X's are all popped.

- We can start a new group (transition from r to p) only when all X's (which count the 0's) have been matched against 1's.

Formal PDA

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where Q, Σ, q_0, and F have their meanings from FA.

- Γ = stack alphabet.
- Z_0 in Γ = start symbol = the one symbol on the stack initially.

- δ = transition function takes a state, an input symbol (or ϵ), and a stack symbol and gives you a finite number of choices of:

  1. A new state (possibly the same).

  2. A string of stack symbols to replace the top stack symbol.

Instantaneous Descriptions (ID's)

For a FA, the only thing of interest about the FA is its state. For a PDA, we want to know its state and the entire content of its stack.

- It is also convenient to maintain a fiction that there is an input string waiting to be read.

- Represented by an ID (q, w, α), where q = state, w = waiting input, and α = stack, top left.

Moves of the PDA

If $\delta(q, a, X)$ contains (p, α), then $(q, aw, X\beta) \vdash (p, w, \alpha\beta)$.

- Extend to $\overset{*}{\vdash}$ to represent 0, 1, or many moves.

- Subscript by name of the PDA, if necessary.

- Input string w is accepted if $(q_0, w, Z_0) \overset{*}{\vdash} (p, \epsilon, \gamma)$ for any accepting state p and any stack string γ.

- $L(P)$ = set of strings accepted by P.

Example

$(p, 0110011, Z_0) \vdash (q, 110011, XZ_0) \vdash (r, 10011, Z_0) \vdash (r, 0011, Z_0) \vdash (p, 0011, Z_0) \vdash (q, 011, XZ_0) \vdash (q, 11, XXZ_0) \vdash (r, 1, XZ_0) \vdash (r, \epsilon, Z_0) \vdash (p, \epsilon, Z_0)$

Acceptance by Empty Stack

Another one of those technical conveniences: when we prove that PDA's and CFG's accept the same languages, it helps to assume that the stack is empty whenever acceptance occurs.

- $N(P)$ = set of strings w such that $(q_0, w, Z_0) \overset{*}{\vdash} (p, \epsilon, \epsilon)$ for some state p.

  - Note p need not be in F.

  - In fact, if we talk about $N(P)$ only, then we need not even specify a set of accepting states.

Example

For our previous example, to accept by empty stack:

1. Add a new transition $\delta(p, \epsilon, Z_0) = \{(p, \epsilon)\}$.

    ❖ That is, when starting to look for a new 0-1 block, the PDA has the option to pop the last symbol off the stack instead.

2. p is no longer an accepting state; in fact, there *are* no accepting states.

Equivalence of Acceptance by Final State and Empty Stack

A language is $L(P_1)$ for some PDA P_1 if and only if it is $N(P_2)$ for some PDA P_2.

- Given $P_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, construct P_2:

    1. Introduce new start state p_0 and new bottom-of-stack marker X_0.

    2. First move of P_2: replace X_0 by Z_0X_0 and go to state q_0. The presence of X_0 prevents P_2 from "accidentally" emptying its stack and accepting when P_1 did not accept.

    3. Then, P_2 simulates P_1; i.e., give P_2 all the transitions of P_1.

    4. Introduce a new state r that keeps popping the stack of P_2 until it is empty.

    5. If (the simulated) P_1 is in an accepting state, give P_2 the additional choice of going to state r on ϵ input, and thus emptying its stack without reading any more input.

- Given $P_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, construct P_1:

    1. Introduce new start state p_0 and new bottom-of-stack marker X_0.

    2. First move of P_1: replace X_0 by Z_0X_0 and go to state q_0.

    3. Introduce new state r for P_1; it is the only accepting state.

    4. P_1 simulates P_2.

    5. If (the simulated) P_1 ever sees X_0, it knows P_2 accepts, so P_1 goes to state r on ϵ input.

Equivalence of CFG's and PDA's

The title says it all.

- We'll show a language L is $L(G)$ for some CFG if and only if it is $N(P)$ for some PDA P.

Only If (CFG to PDA)

Let $L = L(G)$ for some CFG $G = (V, \Sigma, P, S)$.

- Idea: have PDA A simulate LM derivations in G, where a left-sentential form is represented by:

  1. The sequence of input symbols that A has consumed from its input, followed by

  2. A's stack, top leftmost.

- Example: If $(q, abcd, S) \overset{*}{\vdash} (q, cd, ABC)$, then the LSF represented is $abABC$.

Moves of A

- If a terminal a is on top of the stack, then there better be an a waiting on the input. A consumes a from the input and pops it from the stack, if so.

  ✥ The LSF represented doesn't change!

- If a variable B is on top of the stack, then PDA A has a choice of replacing B on the stack by the body of any production with head B.

Formal Construction of A

$A = (\{q\}, \Sigma, V \cup \Sigma, \delta, q, S)$, where δ is defined by:

1. If B is in V, then $\delta(q, \epsilon, B) = \{(q, \alpha) \mid B \to \alpha$ is in $P\}$.

2. If a is in Σ, then $\delta(q, a, a) = \{(q, \epsilon)\}$.

Example

$G = (\{S, A\}, \{0, 1\}, P, S)$, where P consists of $S \to AS \mid \epsilon$; $A \to 0A1 \mid A1 \mid 01$.

- $A = (\{q\}, \{0, 1\}, \{0, 1, A, S\}, \delta, q, S)$, where δ is defined by:

  ✥ $\delta(q, \epsilon, S) = \{(q, AS), (q, \epsilon)\}$

  ✥ $\delta(q, \epsilon, A) = \{(q, 0A1), (q, A1), (q, 01)\}$

  ✥ $\delta(q, 0, 0) = \{(q, \epsilon)\}$

  ✥ $\delta(q, 1, 1) = \{(q, \epsilon)\}$

Only-If Proof (i.e., Grammar $\Rightarrow$ PDA)

- Prove by induction on the number of steps in the derivation $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$ that for any x, $(q, wx, S) \overset{*}{\vdash} (q, x, \beta)$, where
  1. $w\beta = \alpha$.
  2. β is the suffix of α that begins at the leftmost variable ($\beta = \epsilon$ if there is no variable).
- Also prove the converse, that if $(q, wx, S) \overset{*}{\vdash} (q, x, \beta)$, then $S \overset{*}{\Rightarrow} w\beta$.
- Inductive proofs in reader.
- As a consequence, if y is a terminal string, then $S \overset{*}{\Rightarrow} y$ iff $(q, y, S) \overset{*}{\vdash} (q, \epsilon, \epsilon)$, i.e., y is in $L(G)$ iff y is in $N(A)$.

PDA to CFG

Assume $L = N(P)$, where $P = (Q\Sigma, \Gamma, \delta, q_0, Z_0)$.

- Key idea: units of PDA action have the net effect of popping one symbol from the stack, consuming some input, and making a state change.
- The triple $[qZp]$ is a CFG variable that generates exactly those strings w such that P can read w from the input, pop Z (net effect), and go from state q to state p.
  - ✥ More precisely, $(q, w, Z) \overset{*}{\vdash} (p, \epsilon, \epsilon)$.
  - ✥ As a consequence of above, $(q, wx, Z\alpha) \overset{*}{\vdash} (p, x, \alpha)$ for any x and α.
- It's a Zen thing: $[qZp]$ is at once a triple involving states and symbols of P, and yet to the CFG we construct it is a single, indivisible object.
  - ✥ OK; I know that's not a Zen thing, but you get the point.
- Complete proof is in the reader. We'll just give some examples and the idea behind the construction.
- Example: a popping rule, e.g., (p, ϵ) in $\delta(q, a, Z)$.
  - ✥ $[qZp] \rightarrow a$

- A rule that replaces one symbol and state by others, e.g., (p, Y) in $\delta(q, a, Z)$.
  - ✥ For all states r: $[qZr] \rightarrow a[pZr]$
- A rule that replaces one stack symbol by two, e.g., (p, XY) in $\delta(q, a, Z)$.
  - ✥ For all states r and s: $[qZs] \rightarrow a[pXr][rYs]$

Deterministic PDA's

Intuitively: never a choice of move.

- $\delta(q, a, Z)$ has at most one member for any q, a, Z (including $a = \epsilon$).
- If $\delta(q, \epsilon, Z)$ is nonempty, then $\delta(q, a, Z)$ must be empty for all input symbols a.

Why Care?

Parsers, as in YACC, are really DPDA's.

- Thus, the question of what languages a DPDA can accept is really the question of what programming language syntax can be parsed conveniently.

Some Language Relationships

- Acceptance by empty stack is hard for a DPDA.
  - ✥ Once it accepts, it dies and cannot accept any continuation.
  - ✥ Thus, $N(P)$ has the *prefix property*: if w is in $N(P)$, then wx is NOT in $N(P)$ for any $x \neq \epsilon$.
- However, parsers *do* accept by emptying their stack.
  - ✥ Trick: they really process strings followed by a unique endmarker (typically \$) e.g., if they accept $w\$$, they consider w to be a correct program.
- If L is a regular language, then L is a DPDA language.
  - ✥ A DPDA can simulate a DFA, without using its stack (acceptance by final state).
- If L is a DPDA language, then L is a CFL that is *not* inherently ambiguous.
  - ✥ A DPDA yields an unambiguous grammar in the standard construction.