ECE 664
Lecture
23
Avi Kak

Bracket Languages and Pushdown Automata

- Why do we want to learn bracket languages?

  Answer: Because our goal is to demonstrate that we can devise an automaton (along the lines of a dfa) that will accept a context-free language. For reasons that will be clear shortly, this automaton will be called a pushdown automaton.

- For a CF language L on a terminal alphabet T generated by a Chomsky normal form grammar Γ containing n branching productions of the form $X_i \rightarrow Y_i Z_i$, we will consider the following <u>three</u> bracket languages:

  1. $PAR_n(T)$
  2. $L(\Gamma_s)$ — Γ_s is called the separator of Γ
  3. $L(\Delta)$ — Δ is called the spreader of Γ_s

  Strictly speaking, only $PAR_n(T)$ for different n are bracket languages. But it is common to refer to all three by that name.

  Say we are given a set of n brackets:
  $$P = \{ \underset{1}{(}\ \underset{1}{)},\ \underset{2}{(}\ \underset{2}{)},\ \ldots,\ \underset{n}{(}\ \underset{n}{)} \}$$
  $PAR_n(T)$ is composed of every possible word on $T \cup P$ with all possible balanced pairings of all possible brackets in P.

  $P = \{ (,), [,] \}$ $\quad T = \{a, b\}$

  $PAR_2(T) = (), [], ([]), (a)[], (a)[b], ([a])$ ...

  $PAR_n(T)$ is generated by the CF grammar Γ_0: $S \rightarrow a$ for all $a \in T$, $S \rightarrow \underset{i}{(} S \underset{i}{)}$; $i = 1, \ldots, n$, $S \rightarrow SS$, $S \rightarrow 0$

$L(\Gamma_s)$: For any given CF grammar Γ as stated above, we will now define another CF grammar Γ_s in the following manner:

- For each production $X_i \rightarrow Y_i Z_i$ of Γ, Γ_s will have the production
$$X_i \rightarrow \underset{i}{(} Y_i \underset{i}{)} Z_i \qquad i = 1, 2, \ldots, n$$
- Γ_s will have all other productions of Γ unchanged, these being $V \rightarrow a$ for $a \in T$.
- Note that even when Γ is ambiguous (meaning that the same $u \in T^*$ will have more than one leftmost derivation), Γ_s will not be.

Example:

Γ: $S \rightarrow XY$ $\quad S \rightarrow YX$ $\quad Y \rightarrow ZZ$ $\quad X \rightarrow a$ $\quad Z \rightarrow a$

$S \Rightarrow XY \Rightarrow aY \Rightarrow aZZ \Rightarrow aaZ \Rightarrow aaa$
$S \Rightarrow YX \Rightarrow ZZX \Rightarrow aZX \Rightarrow aaX \Rightarrow aaa$ } shows Γ is ambiguous

Γ_s: $S \rightarrow (X)Y$ $\quad S \rightarrow [Y]X$ $\quad Y \rightarrow \{Z\}Z$ $\quad X \rightarrow a$ $\quad Z \rightarrow a$

$S \Rightarrow (X)Y \Rightarrow (a)Y \Rightarrow (a)\{Z\}Z \Rightarrow (a)\{a\}Z \Rightarrow (a)\{a\}a$
$S \Rightarrow [Y]X \Rightarrow [\{Z\}Z]X \Rightarrow [\{a\}Z]X \Rightarrow [\{a\}a]X \Rightarrow [\{a\}a]a$ } shows Γ_s is unambiguous

Γ_s separates the two derivations of Γ. The brackets in the final words allow their respective derivation trees to be recovered.

- Obviously, $L(\Gamma) = Er_p(L(\Gamma_s))$ ← see Lecture 22 for definition of Er_p
- Also, obviously, $L(\Gamma_s) \subseteq PAR_n(T)$

$L(\Delta)$: We will now define a <u>regular grammar</u> Δ from the CF grammar Γ_s: (actually a right-linear grammar)

- The productions of Δ are
  a) all productions $V \rightarrow a$ of Γ_s. (These are the same as $V \rightarrow a$ of Γ)
  b) all productions $X_i \rightarrow \underset{i}{(} Y_i$ for every production $X_i \rightarrow \underset{i}{(} Y_i \underset{i}{)} Z_i$ of Γ_s
  c) all productions $V \rightarrow a \underset{i}{)} Z_i$ for every $V \rightarrow a$ in Γ_s and every $X_i \rightarrow \underset{i}{(} Y_i \underset{i}{)} Z_i$ in Γ_s

We can think of (b) and (c) as "spreading out" of the $X_i \rightarrow \underset{i}{(} Y_i \underset{i}{)} Z_i$ of Γ_s.

Example of $L(\Delta)$:

T: $S \to XY \quad X \to ZY \quad X \to a \quad Y \to b \quad Z \to c$

T_s: $S \to (X)Y \quad X \to [Z]Y \quad X \to a \quad Y \to b \quad Z \to c$

Δ: $S \to (X \quad X \to [Z \quad X \to a \quad Y \to b \quad Z \to c$
$X \to a)Y \quad X \to a]Y$
$Y \to b)Y \quad Y \to b]Y$
$Z \to c)Y \quad Z \to c]Y$

The words of $L(\Delta)$ look like:

$S \Rightarrow (X \Rightarrow (a$
$S \Rightarrow (X \Rightarrow (a)Y \Rightarrow (a)b$
$S \Rightarrow (X \Rightarrow ([Z \Rightarrow ([c$
$S \Rightarrow (X \Rightarrow ([Z \Rightarrow ([c]Y \Rightarrow ([c]b$
$\vdots$

- Since Δ is right-linear, $L(\Delta)$ is a regular language
- It is obvious that $L(T_s) \subseteq L(\Delta)$ (see text for formal proof)
- All words of $L(\Delta)$ that have a symmetric and correct placement of brackets must belong to $L(T_s)$. That is

$$L(\Delta) \cap PAR_n(T) = L(T_s)$$

(see text for formal proof)

- Therefore, $L(T) = Er_p\{L(\Delta) \cap PAR_n(T)\}$
- Equivalently, $\boxed{L(T) = Er_p\{R \cap PAR_n(T)\}}$ → Chomsky-Schutzenberger Representation Theorem

R: since $L(\Delta)$ is regular

One of the most significant results in formal languages

An Important Consequence of $L(T_s) = R \cap PAR_n(T)$:

- This equation means that we can think of an automaton for accepting the words of a CF language generated by a separating grammar.
- We can obviously use a dfa to accept a word in R. But how do we impose the constraint imposed by $PAR_n(T)$? The intersection with $PAR_n(T)$ means that we should accept a word of R only if it has a "balanced" placement of all the brackets.
- We can enforce the bracket-symmetry constraint of $PAR_n(T)$ with the help of a pushdown stack (more commonly known as just stack).
- A pushdown stack operates in a last-in-first out (LIFO) manner.
- At each step of computation with a pushdown stack, one or both of the following operations may be carried out:
  1) the symbol at the top of the stack may be read and then ~~discarded~~ discarded (popping the stack)
  2) a new symbol may be pushed into the stack
- In order to use a stack to identify a string as belonging to $PAR_n(T)$, we introduce a special symbol J_i for each pair of brackets $(_i\ )_i$, $i = 1, 2, \ldots, n$.
- As a string belonging to R is scanned left to right, we push J_i into the stack whenever we see $(_i$ and pop the stack eliminating a J_i when we see $)_i$.
- Such an automaton will successfully scan an entire string of Δ and terminate with an empty stack in case the string belongs to $L(T_s)$.

Definition of a Pushdown Automaton:

A pushdown automaton consists of:

1. a set Q of states
2. an initial state $q_1 \in Q$
3. A set $F \subseteq Q$ of acceptance states
4. Tape alphabet A (same as T)
5. Pushdown alphabet Ω
6. The special symbol $\mathbf{0}$
7. A finite set of transitions

A pushdown automaton transition is specified

$$q_i\, a\, U : V\, q_j$$

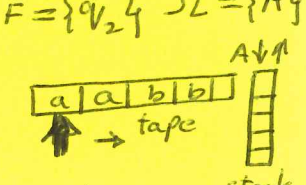
meaning: in state q_i scanning the symbol a with U on top of the stack, move one square to the right, pop the stack removing U, push V into the stack, and enter state q_j

If $a = \mathbf{0}$, no motion to the right
If $U = \mathbf{0}$, no popping of stack
If $V = \mathbf{0}$, no pushing into stack

M accepts $u \in T^*$ if the final state is in F and the stack is empty

Ex.: $Q = \{q_1, q_2\}$ $F = \{q_2\}$ $\Omega = \{A\}$

M: $q_1\, a\, \mathbf{0} : A\, q_1$
$q_1\, b\, A : \mathbf{0}\, q_2$
$q_2\, b\, A : \mathbf{0}\, q_2$



$L(M) = \{a^{[n]} b^{[n]} \mid n > 0\}$