Compiler Design

6 Phases of Compilation

1. Lexical Analysis
2. Syntax Analysis
3. Semantic Analysis
4. Intermediate Code Generation
5. Code Optimization
6. Assembly

Review of Grammars

Grammar - A 4-tuple G = {N, ∑, P, S) where:

1.) N is a finite set of nonterminals
2.) ∑ is a finite set of terminals
3.) P is a finite subset of:
$(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$

P is a set of productions or rules
e.g. (a, b) ∈ P is usually written a->b
4.) S ∈ N, the initial nonterminal

<u>Example</u>:
G = ({E, T, F}, {+, *, (,), a}, P, E)
where P:

```
E -> E + T  |  F
T -> T * F  |  F
F -> ( E )  |  a
```

For example, a derivation tree exists for w = (a + a) * (a + a), so it is an element of the language L(G) (i.e. w ∈ L(G)).

Sentential Form - Form of a grammar G = (N, ∑, P, S), defined recursively:
1.) S is a sentential form
2.) If αβγ is a sentential form and β -> δ ∈ P, then αδγ is also a sentential form.

Sentence - A sentential form that consists only of terminals.

Language - The language generated by a grammar G = (N, ∑, P, S), denoted by L(G), is: {x ∈ ∑* | x is a sentence of G}.

Types of grammars

(1.) Right-Linear - A grammar $G = (N, \sum, P, S)$ is right-linear if every production is of the form $A \rightarrow x\ B$ or $A \rightarrow x$, where $A, B \in N$, $x \in \sum^*$.

(2.) Context-Free - If every production has a single nonterminal on its left-hand side. So, all Right-Linear Grammars are Context-Free; Right-Linear is a more restrictive grammar.

(3.) Context-Sensitive - If every production has the form $\alpha \rightarrow \beta$ such that $|\alpha| \le |\beta|$.

(Here, $|\Delta|$ indicates the length of Δ.)

Grammar Forms

(1.) Proper Form - A grammar G is in proper form if it is

1.) ε-free,
2.) cycle-free, and
3.) it has no symbols that are nonproductive or inaccessible

(See handout for details on converting a CFG into proper form)

(2.) Chomsky Normal Form (CNF) - A grammar G is in CNF if G is ε-free and every non-ε production is of the form $A \rightarrow BC$ or $A \rightarrow a$, where $A, B, C \in N$, $a \in \sum$. Every CFG can be represented in Chomsky Normal Form.

Determining If a String Can Be Produced by a Given Grammar

Cocke-Younger-Kasami (CYK) Algorithm - This algorithm solves the membership problem and works for all CFGs. However, it is $O(|n^3|)$. We are given, as input, a CFG in CNF and an input string x. Note that all CNF derivation trees are binary trees.

-First draw the lower-triangular portion of a matrix n x n (where $n = |\ x\ |$).

-Next, we populate the entries of the table:

$$\begin{aligned} S_{i,j} = {} & (S_{i-1,\ j} * S_{1,\ j+(i-1)}) \cup \\ & (S_{i-2,\ j} * S_{2,\ j+(i-2)}) \cup \\ & \ldots \cup \\ & (S_{1,\ j} * S_{i-1,\ j+1}) \end{aligned}$$

(Examples are given in class; also see handout for details)

Pushdown Automata

E.g.: L = $\{a^n b^n \mid n>0\}$; note that L is non-regular

Initially, we think of using a ***counter*** to solve this problem. With pushdown automata, counter is implemented as a ***stack***.

	counter method	stack method
read an 'a'	ctr++	s.push()
read a 'b'	ctr--	s.pop()
finish	ctr==0	s.empty()

Nondeterministic Pushdown Automata (PDA) - A 7-tuple M = $\{S, \sum, \Gamma, \delta, P_0, Z_0, F)$ where:

1.) S is a finite set of states
2.) $\sum$ is a finite set of input symbols
3.) Γ is a finite set of stack symbols
4.) δ is a function mapping
$S \times (\sum \cup \{\varepsilon\}) \times \Gamma$ into a subset of $S \times \Gamma^*$
5.) $P_0 \in S$, the initial state
6.) $Z_0 \in \Gamma$, the initial stack symbol
7.) $F \subseteq S$, the set of final states

A transition function looks like this:

$$\delta(P, a, z) = (q, \gamma)$$

Or, nondeterministically, like this:

$$\delta(P, a, z) = \{(q, \gamma_1)\ (s, \gamma_2)\}$$

Where different γ values will determine the behavior. For instance:

γ value	action
$\gamma=\varepsilon$	pop
$\gamma = z$	no change
$\gamma = zz$	push z

Inputs are, of course, read from a one-way, read-only input tape (like a finite state automaton).

Configuration (of a PDA) - the state of the PDA at a given instant. It is represented by a triple $(P, W, \alpha) \in S \times \sum^* \times \Gamma^*$, where:

1.) P is the current state.
2.) W is the string of input symbols remaining on the tape (unread input).
3.) α represents the contents of the stack.

Move Relation - For a given PDA M, this is denoted by $\mapsto$; it is defined on pairs of configurations:

$$(P, aw, Z\alpha) \mapsto (q, w, \gamma\alpha) \text{ if } (q, \gamma) \in \delta(P, a, z).$$

The language accepted by a PDA M by final state, denoted by L(M), is

$\{ x \in \sum^* \mid (p0, x, z0) \mapsto^* (q, \varepsilon, \gamma)$, where $q \in F$ and $\gamma \in \Gamma^*\}$; F denotes the set of final states.

The language accepted by a PDA M by empty stack, denoted by $L_e(M)$, is

$\{ x \in \sum^* \mid (p0, x, z0) \mapsto^* (q, \varepsilon, \varepsilon)$, where $q \in S\}$; q is any state.

Example 1. A PDA for $L = \{a^n b^n \mid n \geq 0\}$:

1.	$\delta(P_0, \varepsilon, Z_0) =$	(P_0, ε)	//accept ε
2.	$\delta(P_0, a, Z_0) =$	(P_0, AZ_0)	//1st push
3.	$\delta(P_0, a, A) =$	(P_0, AA)	//continuous push
4.	$\delta(P_0, b, A) =$	(P_1, ε)	//1st pop
5.	$\delta(P_1, b, A) =$	(P_1, ε)	//continuous pop
6.	$\delta(P_1, \varepsilon, Z_0) =$	(P_1, ε)	//accept $a^n b^n$, n>0

We can also create a deterministic PDA for wcw^R, where c is the 'sentinel' value to denote the middle of the string. The more useful ww^R, shown below, must be nondeterministic.

<u>Example 2.</u> A PDA for $L = \{ww^R \mid w \in \{a,b\}^+\}$:

1. $\delta(P_0, a, Z_0) = (P_0, AZ_0)$ //1st push
2. $\delta(P_0, b, Z_0) = (P_0, BZ_0)$ //1st push
3. $\delta(P_0, a, A) = \{(P_0, AA)(P_1, \varepsilon)\}$
4. $\delta(P_0, a, B) = (P_0, AB)$
5. $\delta(P_0, b, A) = (P_0, BA)$
6. $\delta(P_0, b, B) = \{(P_0, BB)(P_1, \varepsilon)\}$ //continuous push or turn around
7. $\delta(P_1, a, A) = (P_1, \varepsilon)$ //pop
8. $\delta(P_1, b, B) = (P_1, \varepsilon)$ //pop
9. $\delta(P_1, \varepsilon, Z_0) = (P_1, \varepsilon)$ //accept

<u>Example 3.</u> A PDA for $L = \{x \in \{a, b\}^* \mid x$ has exactly twice as many a's as b's$\}$

1. $\delta(P_0, a, Z_0) =$ (P_0, AZ_0) //initial push
2. $\delta(P_0, a, A) =$ (P_0, AA) //continuous push
3. $\delta(P_0, b, A) =$ (P_1, ε) //pop 2 A's
4. $\delta(P_1, \varepsilon, A) =$ (P_o, ε) //pop 2 A's
5. $\delta(P_1, \varepsilon, Z_0) =$ (P_o, BZ_0) //handling 'aba'
6. $\delta(P_0, b, Z_0) =$ (P_0, BBZ_0) //initial push
7. $\delta(P_0, b, B) =$ (P_0, BBB) //continuous push
8. $\delta(P_0, a, B) =$ (P_0, ε) //pop 1 B
9. $\delta(P_0, \varepsilon, Z_0) =$ (P_0, ε) //accept

The associated logic behind this solution:
If 1st symbol = 'a': PUSH 1 A for each 'a'
POP 2 A's for each 'b'
If 1st symbol = 'b': PUSH 2 B's for each 'b'
POP 1 B for each 'a'

PDA by final state ≡ PDA by empty stack

$$\begin{tikzcd} M \arrow[r, shift left] & M' \arrow[l, shift left] \end{tikzcd}$$

For every language, if you can find a PDA M that will accept it by final state, you can find an equivalent PDA M' that will accept it by empty stack, and vice versa. This can be shown through simple constructions, but we will skip the formal proof of this.

Equivalence of PDAs and CFLs

Theorem: $\forall$ CFG G, $\exists$ a PDA M such that $L_e(M) = L(G)$.

Algorithm: Let $G = (N, \sum, P, S)$ be a CFG.
Construct the PDA $M = (\{q\}, \sum, N \cup \sum, \delta, q, S, \emptyset)$ where δ:

$(q, \alpha) \in \delta(q, \varepsilon, A)$ if $A \rightarrow \alpha \in P$

$\delta(q, a, a) = (q, \varepsilon)$ $\forall a \in \sum$

<u>Example:</u> Given a CFG $G = (\{E, T, F\}, \{+, *, (,), a\}, P, E)$ where P:

$$E \rightarrow E + T \;|\; F$$
$$T \rightarrow T * F \;|\; F$$
$$F \rightarrow (E) \;|\; a$$

Construct M such that $L_e(M) = L(G)$:
$M = (\{q\}, \{+, +, (,), a\}, \{E, T, F, +, *, (,), a\}, \delta, q, E, \emptyset)$ where δ:

1. $\delta(q, \varepsilon, E) = \{(q, E+T)\ (q, T)\}$
2. $\delta(q, \varepsilon, T) = \{(q, T*F)\ (q, F)\}$
3. $\delta(q, \varepsilon, F) = \{(q, (E))\ (q, a)\}$
4. $\delta(q, +, +) = (q, \varepsilon)$
5. $\delta(q, *, *) = (q, \varepsilon)$
6. $\delta(q, (, ( ) = (q, \varepsilon)$
7. $\delta(q,),)) = (q, \varepsilon)$
8. $\delta(q, a, a) = (q, \varepsilon)$

Determinism

Deterministic CFL – A language is a deterministic CFL if it is accepted by a deterministic PDA.

Deterministic PDA – A PDA $M = \{S, \sum, \Gamma, \delta, P_0, Z_0, F)$ is deterministic if:

1. $\forall\ P \in S,\ a \in \sum,\ Z \in \Gamma$
   $\delta(P, \varepsilon, Z) = \emptyset$ and $|\ \delta(P, a, Z)\ | \leq 1$, and
   (e-move)   (input move)

2. $\forall\ P \in S,\ Z \in \Gamma$
   $\forall\ a \in \sum\ [\delta(P, a, Z) = \emptyset]$ and $|\ \delta(P, \varepsilon, Z)\ | \leq 1$.

Essentially, to show that a given PDA is nondeterministic, we must find a pair of moves such that one is an e-move, one is an input move, and the other values (P and Z) are the same. Also, a transition that yields two moves illustrates nondeterminism.

Deterministic Context-Free Languages

Deterministic context-free languages are those languages that are accepted by deterministic PDAs. The class of deterministic context-free languages is closed under complement, unlike the superclass of nondeterministic context-free languages. The class of context-free languages properly contains the class of deterministic CFLs; nondeterminism gives a PDA more power.