Top-Down Parsing

Lecture Eight

Predictive Parsing

Predictive Parsers

- Predictive Parsers
  - parser can “predict” which production to use By looking at the next few tokens (look ahead).
  - It uses a restricted form of grammar (LL(k) grammars)
  - LL(k) stands for **L**eft to right scan and **L**eft most derivation for k look ahead tokens.
  - k usually =1. therefore, it usually called LL(1) Parser
  - At each step, only one choice of production.
  - No backtracking(grammar is deterministic).

Left Recursion

Left recursion is used to make operations left associative.

Simple immediate left recursion:

$S \rightarrow S\ \alpha \mid \beta$

Left Factoring

Left factoring is required when two or more grammar rule choices share a common prefix string.

$E \rightarrow T + E \mid T$

$T \rightarrow int \mid int * T \mid (E)$

- This grammar is not acceptable for LL(1) Parsing because it is hard to predict the right production
  - For T : two productions start with int
  - For E : two productions start with T
- We need to left-factor the grammar(Nondeterministic grammar to deterministic)

LL(1) Parsing Table

$E \rightarrow T\ X$ $\quad\quad$ $X \rightarrow + E \mid \varepsilon$

$T \rightarrow (E) \mid int\ Y$ $\quad\quad$ $Y \rightarrow * T \mid \varepsilon$

Input Tokens (Terminals)

	int	*	+	(	)	$
E	E →T X			E →T X		
X			X → + E		X → ε	X → ε
T	T →int Y			T →(E)		
Y		Y → * T	Y →ε		Y →ε	Y →ε

Left most non Terminal

RHS of Production

[E, int] entry : current non-terminal is E and next input is int→ use production E → T X

[Y,+] current non-terminal is Y and current token is +, get rid of Y Y →epsilon

[E,*] entry –"There is no way to derive a string starting with * from non-terminal E error

LL(1) Parsing Table

- Use stack instead of recursive function in recursive descent.
- $ marks end of input or bottom of the stack.
- Push start symbol.
- While (!emptyStack){

  case stack of

  <nonTerminal, rest> : if exist in Parsing table
  replace with RHS of production
  else error

  <Terminal, rest> : if t == *input++ Match terminal
  else error }

Reject on reaching error state

Accept if input string and stack t become empty

LL(1) Parsing Table

Stack	Input	Action
$ E	int*int$	Replace **E →T X**
$ X T	int*int$	Replace **T→int Y**
$ X Y int	int*int$	Match(Pop int)
$ X Y	*int$	Replace **Y → * T**
$ X T *	*int$	Match
$ X T	int$	Replace **T→int Y**
$ X Y int	int$	Match
$ X Y	$	Replace **Y→ε**
$ X	$	Replace **X→ε**
$	$	Accept

Input → int *int $

E →T X
T →(E) | int Y
X → + E | ε
Y → * T | ε

	int	*	+	(	)	$
E	E → T X			E → T X		
X			X → + E		X → ε	X → ε
T	T → int Y			T → (E)		
Y		Y → * T	Y →ε		Y →ε	Y →ε

LL(1) Parsing Table

- LL(l) parsing table : a two-dimensional array *M[N, T]*.
- *N:nonterminal , Terminal*
- We add production choices to this table according to the following rules for production rule $A \rightarrow \alpha$:

1. There is a derivation $\alpha =>^* a$ X, add $A \rightarrow \alpha$ to the table entry *M[A, a]*.

2. There are derivations $\alpha \rightarrow \varepsilon$ and S=>* AaY.

S: start Symbol add $A \rightarrow \alpha$ to the table entry *M[A, a]*.

These rules are difficult to implement so the First and Follow sets are used.

First Sets

- If X is a terminal or ε, then First(X) = {X}.
- If X is a nonterminal,

$$\mathbf{X \rightarrow X_1 X_2 \ldots X_n}$$

  - {First(X_1)–{ε}}⊂ First(X)
  - While (**ε** ∈ First(Xi))

{

{First(X_{i+1})–{ε}}⊂ First(X)

}

  - If **ε** ∈ {First(X_1), .., First(X_n)} then **ε** ∈ First(X).

First Sets

E →T X

X → + E | ε

T →(E) | int Y

Y → * T | ε

First of Terminal

First(+)={+}

First(*)={*}

First(( )={(}

First( ))={)}

First(int)={int}

First of nonterminal

First(X)={+, ε}

First(Y)={*, ε}

First(T)={(,int}

First(E)= First (T)={(,int}

Follow Sets

Nonterminal A, the set Follow(A):

1. If A is the start symbol, then \$ is in Follow(A).

2. If there is a production $B \rightarrow \alpha\ A\ \gamma$, then First($\gamma$)–{$\varepsilon$} is in Follow($A$).

3. If there is a production $B \rightarrow \alpha\ A\ \gamma$ such that ε is in First(γ), then Follow(A) contains Follow(B).

Follow Sets

E →T X

X → + E | ε

T →(E) | int Y

Y → * T | ε

Follow of Terminal

Follow(+)={(,int}

Follow(*)={(,int}

Follow(()={(,int}

Follow())={+,),$)}

Follow(int)={*, +,),$}

Follow of nonterminal

Follow(E)= {$,)}

Follow(X)= {$,)}

Follow(T)= {+,),$}

Follow(Y)={+,),$}

LL(1) Parsing Table

for each production choice $A \rightarrow \alpha$:

- For every token a in First(α),

  add $A \rightarrow \alpha$ to the entry `M[A,a]`.

- If ε is in First(α), then for every element `x` in Follow(`A`),

  add $A \rightarrow \alpha$ to the entry `M[A,x]`.

LL(1) Parsing Table

$E \rightarrow T\ X$ $\qquad\qquad$ $X \rightarrow + E \mid \varepsilon$

$T \rightarrow (E) \mid int\ Y$ $\qquad\qquad$ $Y \rightarrow * T \mid \varepsilon$

	int	*	+	(	)	$
E	E →T X			E →T X		
X			X → + E		X → ε	X → ε
T	T →int Y			T →(E)		
Y		Y → * T	Y →ε		Y →ε	Y →ε

LL(1)Grammar

- Grammar is not LL(1) when:
  - Left factored grammar
  - Left recursive grammar
  - Ambiguous grammar
  - Multiple entry in LL(1) Parsing Table.