Lecture 19

LL(1) Table Construction

Here now is the algorithm to construct a predictive parsing table.

1. For each production $A \rightarrow \alpha$

   1. for each terminal a in $\text{FIRST}(\alpha)$, add $A \rightarrow \alpha$ to $M[A,a]$.

   2. If ϵ is in $\text{FIRST}(\alpha)$, add $A \rightarrow \alpha$ to $M[A,b]$ for each terminal b in $\text{FOLLOW}(A)$. If ϵ is in $\text{FIRST}(\alpha)$, and \$ is in $\text{FOLLOW}(A)$, add $A \rightarrow \alpha$ to $M[A,\$]$.

2. Make each undefined entry of M be error.

Let us apply the algorithm to the expression grammar. Since $\text{FIRST}(TE') = \text{FIRST}(T) = \{ (, \underline{\text{id}} \}$, the production $E \rightarrow TE'$ cause $M[E,(]$ and $M[E,\underline{\text{id}}]$ to get $E \rightarrow TE'$. The production $E' \rightarrow +TE'$ causes $M[E',+]$ to get $E' \rightarrow +TE'$. The production $E' \rightarrow \epsilon$ causes $M[E',)]$ and $M[E',\$]$ to get $E' \rightarrow \epsilon$ since $\text{FOLLOW}(E') = \{), \$ \}$. And so on. The final parsing table produced is:

	id	+	*	(	)	$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow$ id			$F \rightarrow (E)$		

Left Factoring

Consider the grammar

E → T + E | T
T → int | int * T | (E)

It is impossible to predict because for T, two productions start with int. For E, it is not clear how to predict; the two productions start with the non-terminal T. A grammar must be *left factored* before use for predictive parsing. The procedure to left-factor a grammar is as follows:

If a ¹ e, replace all productions

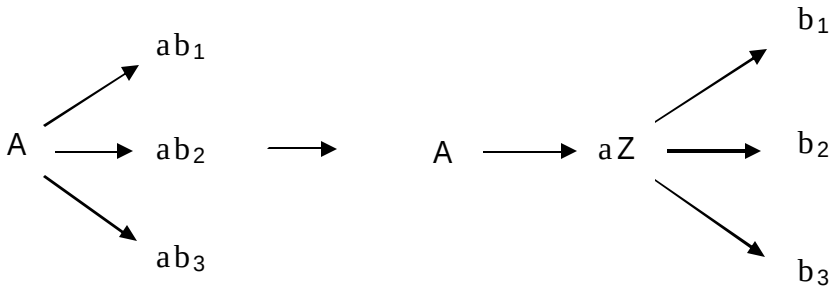
$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid .... \mid \alpha\beta_n \mid \gamma$$

with

$$A \rightarrow \alpha Z \mid \gamma$$
$$Z \rightarrow \beta_1 \mid \beta_2 \mid .... \mid \beta_n$$

where Z is a new non-terminal

A graphical explanation:



Example: consider following fragment of expression grammar

```
Factor ?    id
       |    id [ ExprList ]
       |    id ( ExprList )
```

After left factoring, the grammar becomes

```
Factor  ?   id Args
Args    ?   [ ExprList ]
        |   ( ExprList )
        |   e
```

Given a CFG that does not meet the LL(1) condition, it is *undecidable* whether or not an equivalent LL(1) grammar exists.