Parsing

Part IV

Building the Predictive Parsing Table

Assume we're looking for an A

  i.e., A is on the stack top.

Assume b is the current input symbol.

If $A \to \alpha$ is a rule and b is in FIRST(α)

  then expand A using the $A \to \alpha$ rule!

What if ε is in FIRST(α)? [i.e., $\alpha \Rightarrow^* \varepsilon$]

  If b is in FOLLOW(A)

    then expand A using the $A \to \alpha$ rule!

If ε is in FIRST(α) and \$ is the current input symbol

  then if \$ is in FOLLOW(A)

    then expand A using the $A \to \alpha$ rule!

Constructing LL(1) Parsing Table -- Algorithm

- for each production rule $A \rightarrow \alpha$ of a grammar G
  - for each terminal a in FIRST(α)
    ➔ add $A \rightarrow \alpha$ to M[A,a]
  - If ε in FIRST(α)
    ➔ for each terminal a in FOLLOW(A) add $A \rightarrow \alpha$ to M[A,a]
  - If ε in FIRST(α) and \$ in FOLLOW(A)
    ➔ add $A \rightarrow \alpha$ to M[A,\$]

- All other undefined entries of the parsing table are error entries.

Example: The "Dangling Else" Grammar

1.	S	$\rightarrow$	<u>i</u> E <u>t</u> S S'
2.	S	$\rightarrow$	<u>o</u>
3.	S'	$\rightarrow$	<u>e</u> S
4.	S'	$\rightarrow$	ε
5.	E	$\rightarrow$	<u>b</u>

$\leftarrow$

1.	S	$\rightarrow$	<u>if</u> E <u>then</u> S S'
2.	S	$\rightarrow$	<u>otherStmt</u>
3.	S'	$\rightarrow$	<u>else</u> S
4.	S'	$\rightarrow$	ε
5.	E	$\rightarrow$	<u>boolExpr</u>

<u>i</u> <u>b</u> <u>t</u> <u>i</u> <u>b</u> <u>t</u> <u>o</u> <u>e</u> <u>o</u> $\Leftarrow$ "if b then if b then otherStmt else otherStmt"

Example: The “Dangling Else” Grammar

1.	S	$\rightarrow \underline{\mathtt{i}}\ E\ \underline{\mathtt{t}}\ S\ S'$
2.	S	$\rightarrow \underline{\mathtt{o}}$
3.	S'	$\rightarrow \underline{\mathtt{e}}\ S$
4.	S'	$\rightarrow \varepsilon$
5.	E	$\rightarrow \underline{\mathtt{b}}$

$\underline{i}\ \underline{b}\ \underline{t}\ \underline{i}\ \underline{b}\ \underline{t}\ \underline{o}\ \underline{e}\ \underline{o}$

Example: The "Dangling Else" Grammar

1.	S	→	<u>i</u> E <u>t</u> S S'
2.	S	→	<u>o</u>
3.	S'	→	<u>e</u> S
4.	S'	→	ε
5.	E	→	<u>b</u>

<u>i</u> <u>b</u> <u>t</u> <u>i</u> <u>b</u> <u>t</u> <u>o</u> <u>e</u> <u>o</u>

FIRST(S) = { <u>i</u>, <u>o</u> }
FIRST(S') = { <u>e</u>, ε }
FIRST(E) = { <u>b</u> }

FOLLOW(S) = { <u>e</u>, $ }
FOLLOW(S') = { <u>e</u>, $ }
FOLLOW(E) = { <u>t</u> }

Example: The "Dangling Else" Grammar

1. $S \rightarrow$ i E t S S'
2. $S \rightarrow$ o
3. $S' \rightarrow$ e S
4. $S' \rightarrow \varepsilon$
5. $E \rightarrow$ b

Look at Rule 1: $S \rightarrow$ i E t S S'
If we are looking for an S
and the next symbol is in FIRST(i E t S S')...
Add that rule to the table

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, $ }
FOLLOW(S') = { e, $ }
FOLLOW(E) = { t }

	o	b	e	i	t	$
S						
S'						
E						

Example: The "Dangling Else" Grammar

1. $S \rightarrow \underline{\texttt{i}}\ E\ \underline{\texttt{t}}\ S\ S'$
2. $S \rightarrow \underline{\texttt{o}}$
3. $S' \rightarrow \underline{\texttt{e}}\ S$
4. $S' \rightarrow \varepsilon$
5. $E \rightarrow \underline{\texttt{b}}$

Look at Rule 1: $S \rightarrow \underline{\texttt{i}}\ E\ \underline{\texttt{t}}\ S\ S'$

If we are looking for an S

and the next symbol is in FIRST($\underline{\texttt{i}}\ E\ \underline{\texttt{t}}\ S\ S'$)...

Add that rule to the table

i b t i b t o e o

FIRST(S) = { $\underline{\texttt{i}}$, $\underline{\texttt{o}}$ }

FIRST(S') = { $\underline{\texttt{e}}$, ε }

FIRST(E) = { $\underline{\texttt{b}}$ }

FOLLOW(S) = { $\underline{\texttt{e}}$, \$ }

FOLLOW(S') = { $\underline{\texttt{e}}$, \$ }

FOLLOW(E) = { $\underline{\texttt{t}}$ }

	o	b	e	i	t	\$
S				$S \rightarrow \underline{\texttt{i}}E\underline{\texttt{t}}SS'$		
S'						
E						

Example: The "Dangling Else" Grammar

1. S → i E t S S'
2. S → o
3. S' → e S
4. S' → ε
5. E → b

Look at Rule 2: S → o
If we are looking for an S
and the next symbol is in FIRST(o)...
Add that rule to the table

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, $ }
FOLLOW(S') = { e, $ }
FOLLOW(E) = { t }

	o	b	e	i	t	$
S	S → o			S → iEtSS'		
S'						
E						

Example: The "Dangling Else" Grammar

1.	S	→	i E t S S'
2.	S	→	o
3.	S'	→	e S
4.	S'	→	ε
5.	E	→	b

Look at Rule 5: E → b
If we are looking for an E
and the next symbol is in FIRST(b)...
Add that rule to the table

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, $ }
FOLLOW(S') = { e, $ }
FOLLOW(E) = { t }

	o	b	e	i	t	$
S	S → o			S → iEtSS'		
S'						
E		E → b				

Example: The "Dangling Else" Grammar

1. $S \rightarrow \underline{i}\ E\ \underline{t}\ S\ S'$
2. $S \rightarrow \underline{o}$
3. $S' \rightarrow \underline{e}\ S$
4. $S' \rightarrow \varepsilon$
5. $E \rightarrow \underline{b}$

Look at Rule 3: $S' \rightarrow \underline{e}\ S$

If we are looking for an S'

and the next symbol is in FIRST($\underline{e}$ S)...

Add that rule to the table

i b t i b t o e o

FIRST(S) = { $\underline{i}$, $\underline{o}$ }

FIRST(S') = { $\underline{e}$, ε }

FIRST(E) = { $\underline{b}$ }

FOLLOW(S) = { $\underline{e}$, \$ }

FOLLOW(S') = { $\underline{e}$, \$ }

FOLLOW(E) = { $\underline{t}$ }

	o	b	e	i	t	$
S	$S \rightarrow \underline{o}$			$S \rightarrow \underline{i}E\underline{t}SS'$		
S'			$S' \rightarrow \underline{e}\ S$			
E		$E \rightarrow \underline{b}$				

Example: The "Dangling Else" Grammar

1. $S \rightarrow$ i E t S S'
2. $S \rightarrow$ o
3. $S' \rightarrow$ e S
4. $S' \rightarrow \varepsilon$
5. $E \rightarrow$ b

Look at Rule 4: $S' \rightarrow \varepsilon$

If we are looking for an S'

and $\varepsilon \in$ FIRST(rhs)...

Then if $ $\in$ FOLLOW(S')...

Add that rule under $

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, $ }
FOLLOW(S') = { e, $ }
FOLLOW(E) = { t }

	o	b	e	i	t	$
S	$S \rightarrow$ o			$S \rightarrow$ iEtSS'		
S'			$S' \rightarrow$ e S			
E		$E \rightarrow$ b				

Example: The "Dangling Else" Grammar

1. $S \rightarrow \underline{i}\ E\ \underline{t}\ S\ S'$
2. $S \rightarrow \underline{o}$
3. $S' \rightarrow \underline{e}\ S$
4. $S' \rightarrow \varepsilon$
5. $E \rightarrow \underline{b}$

Look at Rule 4: $S' \rightarrow \varepsilon$

If we are looking for an S'

and $\varepsilon \in$ FIRST(rhs)...

Then if $\$ \in$ FOLLOW(S')...

Add that rule under $

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, $ }
FOLLOW(S') = { e, $ }
FOLLOW(E) = { t }

	o	b	e	i	t	$
S	$S \rightarrow \underline{o}$			$S \rightarrow \underline{i}E\underline{t}SS'$		
S'			$S' \rightarrow \underline{e}\ S$			$S' \rightarrow \varepsilon$
E		$E \rightarrow \underline{b}$				

Example: The "Dangling Else" Grammar

1.	S	$\rightarrow$ i E t S S'
2.	S	$\rightarrow$ o
3.	S'	$\rightarrow$ e S
4.	S'	$\rightarrow$ ε
5.	E	$\rightarrow$ b

Look at Rule 4: S' $\rightarrow$ ε

If we are looking for an S'

and $\varepsilon \in$ FIRST(rhs)...

Then if e $\in$ FOLLOW(S')...

Add that rule under e

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, $ }
FOLLOW(S') = { e, $ }
FOLLOW(E) = { t }

	o	b	e	i	t	$
S	S $\rightarrow$ o			S $\rightarrow$ iEtSS'		
S'			S' $\rightarrow$ e S S' $\rightarrow$ ε			S' $\rightarrow$ ε
E		E $\rightarrow$ b				

Example: The “Dangling Else” Grammar

1. $S \rightarrow \underline{\texttt{i}}\ E\ \underline{\texttt{t}}\ S\ S'$
2. $S \rightarrow \underline{\texttt{o}}$
3. $S' \rightarrow \underline{\texttt{e}}\ S$
4. $S' \rightarrow \varepsilon$
5. $E \rightarrow \underline{\texttt{b}}$

CONFLICT!

Two rules in one table entry.

The grammar is not LL(1)!

i b t i b t o e o

FIRST(S) = { i, o }
FIRST(S') = { e, ε }
FIRST(E) = { b }

FOLLOW(S) = { e, \$ }
FOLLOW(S') = { e, \$ }
FOLLOW(E) = { t }

	o	b	e	i	t	\$
S	$S \rightarrow \texttt{o}$			$S \rightarrow \texttt{i}E\texttt{t}SS'$		
S'			$S' \rightarrow \texttt{e}\ S$ $S' \rightarrow \varepsilon$			$S' \rightarrow \varepsilon$
E		$E \rightarrow \texttt{b}$				

Algorithm to Build the Table

Input: Grammar G
Output: Parsing Table, such that TABLE[A,b] = Rule to use or "ERROR/Blank"

```
Compute FIRST and FOLLOW sets
for each rule A→α do
  for each terminal b in FIRST(α) do
    add A→α to TABLE[A,b]
  endFor
  if ε is in FIRST(α) then
    for each terminal b in FOLLOW(A) do
      add A→α to TABLE[A,b]
    endFor
    if $ is in FOLLOW(A) then
      add A→α to TABLE[A,$]
    endIf
  endIf
endFor
TABLE[A,b] is undefined?  Then set TABLE[A,b] to "error"
TABLE[A,b] is multiply defined?
```

The algorithm fails!!! Grammar G is not LL(1)!!!

Predictive Parsing Example

$$
\begin{aligned}
E &\rightarrow T\,E' \\
E' &\rightarrow +\,T\,E' \mid \varepsilon \\
T &\rightarrow F\,T' \\
T' &\rightarrow *\,F\,T' \mid \varepsilon \\
F &\rightarrow (\,E\,) \mid \underline{\mathrm{id}}
\end{aligned}
$$

FIRST(F) = FIRST(T) = FIRST(E) = { (, id }

FIRST(E′)={+, ε}

FIRST(T′)={*, ε}

FOLLOW(E)=FOLLOW(E′) = {), $}

FOLLOW(T)=FOLLOW(T′)={+,), $}

FOLLOW(F) = {+, *,), $}

A Grammar which is not LL(1)

If the parsing table of a grammar contains more than one production rule then it is not a LL(1) grammar

$S \rightarrow i\ C\ t\ S\ E \mid a$
$E \rightarrow e\ S \mid \varepsilon$
$C \rightarrow b$

FOLLOW(S) = { $,e }
FOLLOW(E) = { $,e }
FOLLOW(C) = { t }

FIRST(iCtSE) = {i}
FIRST(a) = {a}
FIRST(eS) = {e}
FIRST(ε) = {ε}
FIRST(b) = {b}

	a	**b**	**e**	**i**	**t**	**$**
S	$S \rightarrow a$			$S \rightarrow iCtSE$		
E			$E \rightarrow e\ S$ $E \rightarrow \varepsilon$			$E \rightarrow \varepsilon$
C		$C \rightarrow b$				

two production rules for M[E,e]

Problem ➔ ambiguity

A Grammar which is not LL(1) (cont.)

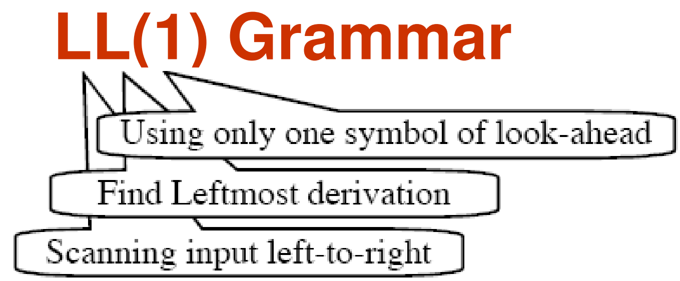
- What do we have to do it if the resulting parsing table contains multiply defined entries?
  - If we didn't eliminate left recursion, eliminate the left recursion in the grammar.
  - If the grammar is not left factored, we have to left factor the grammar.
  - If its (new grammar's) parsing table still contains multiply defined entries, that grammar is ambiguous or it is inherently not a LL(1) grammar.
- A left recursive grammar cannot be a LL(1) grammar.
  - $A \rightarrow A\alpha \mid \beta$

    ➔ any terminal that appears in FIRST(β) also appears FIRST($A\alpha$) because $A\alpha \Rightarrow \beta\alpha$.

    ➔ If β is ε, any terminal that appears in FIRST(α) also appears in FIRST($A\alpha$) and FOLLOW(A).
- A grammar is not left factored, it cannot be a LL(1) grammar
  - $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$

    ➔any terminal that appears in FIRST($\alpha\beta_1$) also appears in FIRST($\alpha\beta_2$).
- An ambiguous grammar cannot be a LL(1) grammar.

Properties of LL(1) Grammars



- A grammar G is LL(1) if and only if the following conditions hold for two distinctive production rules $A \rightarrow \alpha$ and $A \rightarrow \beta$

1. Both α and β cannot derive strings starting with same terminals.

2. At most one of α and β can derive to ε.

3. If β can derive to ε, then α cannot derive to any string starting with a terminal in FOLLOW(A).

Error Recovery

We have an error whenever...

- Stacktop is a terminal, but stacktop ≠ input symbol
- Stacktop is a nonterminal but TABLE[A,b] is empty

Options

1. Skip over input symbols, until we can resume parsing
   Corresponds to ignoring tokens
2. Pop stack, until we can resume parsing
   Corresponds to inserting missing material
3. Some combination of 1 and 2
4. "Panic Mode" - Use Synchronizing tokens
   - Identify a set of synchronizing tokens.
   - Skip over tokens until we are positioned on a synchronizing token.
   - Pop stack until we can resume parsing.

Error Recovery: Skip Input Symbols

Example:

Decided to use rule

$S \rightarrow$ **IF** E **THEN** S **ELSE** S **END**

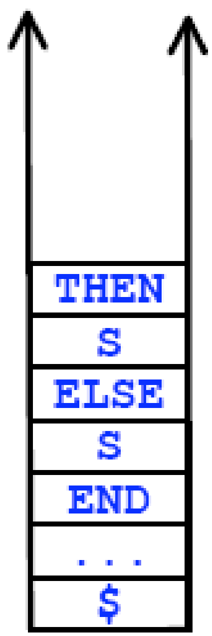
Stack tells us what we are expecting next in the input.

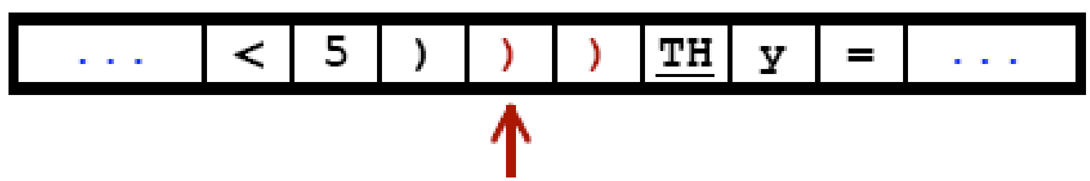
We've already gotten **IF** and E

Assume there are extra tokens in the input.

```
if (x<5))) then y = 7; ...
```

A syntax error occurs here.





We want to skip tokens until we can resume parsing.

Error Recovery: Pop The Stack

Example:

Decided to use rules

S → `IF` E `THEN` S `ELSE` S `END`

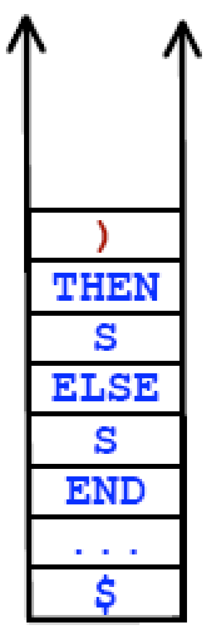
E → `(` E `)`

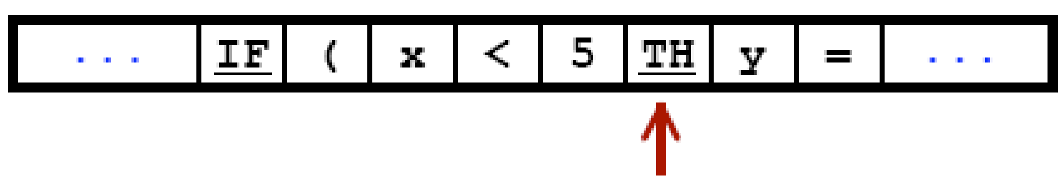
We've already gotten `if ( ` E

Assume there are missing tokens.

```
if (x < 5 then y = 7;...
```

A syntax error occurs here.





We want to pop the stack until we can resume parsing.

Error Recovery: Panic Mode

The "*Synchronizing Set*" of tokens

... is determined by the compiler writer beforehand

Example: { SEMI-COLON, RIGHT-BRACE }

Skip input symbols until we find something in the synchronizing set.

Idea:

Look at the non-terminal on the stack top.

Choose the synchronizing set based on this non-terminal.

Assume A is on the stack top

Let SynchSet = FOLLOW(A)

Skip tokens until we see something in FOLLOW(A)

Pop A from the stack.

Should be able to keep going.

Idea:

Look at the non-terminals in the stack (e.g., A, B, C, ...)

Include FIRST(A), FIRST(B), FIRST(C), ... in the SynchSet.

Skip tokens until we see something in FIRST(A), FIRST(B), FIRST(C), ...

Pop stack until **NextToken** ∈ FIRST(**NonTerminalOnStackTop**)

Error Recovery - Table Entries

Each blank entry in the table indicates an error.
Tailor the error recovery for each possible error.
Fill the blank entry with an error routine.
 The error routine will tell what to do.

Example:

	id	SEMI	RPAREN	LPAREN	...	$
E			E4			
E'			E5			
...						

Choose the SynchSet based on the particular error

Error-Handling Code

```
...
E4:
          SynchSet = { SEMI, IF, THEN }
          SkipTokensTo (SynchSet)
          Print ("Unexpected right paren")
          Pop stack
          break
E5:       ...
...
```

Panic-Mode Error Recovery - Example

$S \rightarrow AbS \mid e \mid \varepsilon$
$A \rightarrow a \mid cAd$

FOLLOW(S)={\$}
FOLLOW(A)={b,d}

	a	**b**	**c**	**d**	**e**	**\$**
S	$S \rightarrow AbS$	*sync*	$S \rightarrow AbS$	*sync*	$S \rightarrow e$	$S \rightarrow \varepsilon$
A	$A \rightarrow a$	*sync*	$A \rightarrow cAd$	*sync*	*sync*	*sync*

stack	**input**	**output**
\$S	aab\$	$S \rightarrow AbS$
\$SbA	aab\$	$A \rightarrow a$
\$Sba	aab\$	
\$Sb	ab\$	Error: missing b, inserted
\$S	ab\$	$S \rightarrow AbS$
\$SbA	ab\$	$A \rightarrow a$
\$Sba	ab\$	
\$Sb	b\$	
\$S	\$	$S \rightarrow \varepsilon$
\$	\$	accept

stack	**input**	**output**
\$S	ceadb\$	$S \rightarrow AbS$
\$SbA	ceadb\$	$A \rightarrow cAd$
\$SbdAc	ceadb\$	
\$SbdA	eadb\$	Error:unexpected e (illegal A)
(Remove all input tokens until first b or d, pop A)		
\$Sbd	db\$	
\$Sb	b\$	
\$S	\$	$S \rightarrow \varepsilon$
\$	\$	accept

Phrase-Level Error Recovery

- Each empty entry in the parsing table is filled with a pointer to a special error routine which will take care that error case.
- These error routines may:
  - change, insert, or delete input symbols.
  - issue appropriate error messages
  - pop items from the stack.
- We should be careful when we design these error routines, because we may put the parser into an infinite loop.