CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

Quick Review of Last Lecture

- Writing a Grammar
  - Left Recursion Elimination Examples
  - Left Factoring
- Top-Down Parsing
  - FIRST Set, FOLLOW Set and examples
  - LL(1) Grammar and examples

Using FIRST and FOLLOW in a Recursive-Descent Parser

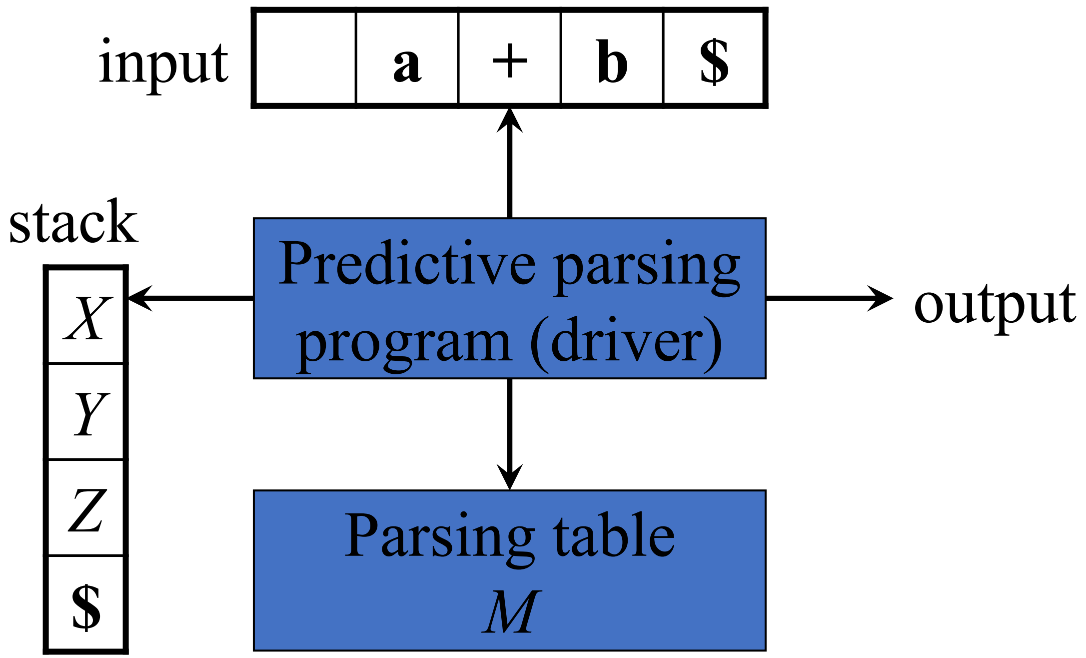
expr → *term rest*
rest → + *term rest*
| **-** *term rest*
| ε
term → **id**

procedure *rest*();
begin
  if *lookahead* in FIRST(+ *term rest*) **then**
    match(**'+'**); *term*(); *rest*()
  else if *lookahead* in FIRST(**-** *term rest*) **then**
    match(**'-'**); *term*(); *rest*()
  else if *lookahead* in FOLLOW(*rest*) **then**
    return
  else error**()**
end;

where	FIRST(+ *term rest*) = { + }
	FIRST(**-** *term rest*) = { **-** }
	FOLLOW(*rest*) = { **$** }

Non-Recursive Predictive Parsing: Table-Driven Parsing

- Given an LL(1) grammar $G = (N, T, P, S)$ construct a table $M[A, a]$ for $A \in N$, $a \in T$ and use a *driver program* with a *stack*



Constructing an LL(1) Predictive Parsing Table

```
for each production A → α {
    for each a ∈ FIRST(α) {
        add A → α to M[A, a]
    }
    if ε ∈ FIRST(α) {
        for each b ∈ FOLLOW(A) {
            add A → α to M[A, b]
        }
    }
}
Mark each undefined entry in M error
```

Example Table

$$E \rightarrow T\,E'$$
$$E' \rightarrow +\,T\,E' \mid \varepsilon$$
$$T \rightarrow F\,T'$$
$$T' \rightarrow *\,F\,T' \mid \varepsilon$$
$$F \rightarrow (\,E\,) \mid \mathbf{id}$$

$A \rightarrow \alpha$	FIRST(α)	FOLLOW(A)
$E \rightarrow T\,E'$	**(id**	**$)**
$E' \rightarrow +\,T\,E'$	+	**$)**
$E' \rightarrow \varepsilon$	ε	
$T \rightarrow F\,T'$	**(id**	+ **$)**
$T' \rightarrow *\,F\,T'$	*	+ **$)**
$T' \rightarrow \varepsilon$	ε	
$F \rightarrow (\,E\,)$	**(**	* + **$)**
$F \rightarrow \mathbf{id}$	**id**	

	id	+	*	**(**	**)**	**$**
E	$E \rightarrow T\,E'$			$E \rightarrow T\,E'$		
E'		$E' \rightarrow +\,T\,E'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow F\,T'$			$T \rightarrow F\,T'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *\,F\,T'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$			$F \rightarrow (\,E\,)$		

LL(1) Grammars are Unambiguous

Ambiguous grammar

$S \rightarrow \mathbf{i}\ E\ \mathbf{t}\ S\ S' \mid \mathbf{a}$

$S' \rightarrow \mathbf{e}\ S \mid \varepsilon$

$E \rightarrow \mathbf{b}$

$A \rightarrow \alpha$	FIRST(α)	FOLLOW(A)
$S \rightarrow \mathbf{i}\ E\ \mathbf{t}\ S\ S'$	**i**	**e $**
$S \rightarrow \mathbf{a}$	**a**	
$S' \rightarrow \mathbf{e}\ S$	**e**	**e $**
$S' \rightarrow \varepsilon$	ε	
$E \rightarrow \mathbf{b}$	**b**	**t**

Error: duplicate table entry

	a	**b**	**e**	**i**	**t**	**$**
S	$S \rightarrow \mathbf{a}$			$S \rightarrow \mathbf{i}\ E\ \mathbf{t}\ S\ S'$		
S'			$S' \rightarrow \varepsilon$ $S' \rightarrow \mathbf{e}\ S$			$S' \rightarrow \varepsilon$
E		$E \rightarrow \mathbf{b}$				

Predictive Parsing Program (Driver)

```
read w$ into the input buffer; // w is the input
push($); push(S);
a = lookahead;              // a is the first symbol of w
X = pop();
while ( X ≠ $ ) {
    if ( X = a ) {a = lookahead;}        // a is next symbol;
    else if ( X is a terminal ) error();
    else if ( M [X, a] is an error entry ) error();
    else if ( M[X, a] = X → Y1Y2 …Yk ) {
        output the production X → Y1Y2 …Yk ;
        push (Yk); push(Yk–1) , ... , push(Y1);
    }
    X = pop();
}
```

Example: Moves of table-driven parsing on input **id + id * id**

	id	+	*	**$**
E	$E \rightarrow TE'$			
E'		$E' \rightarrow +TE'$		$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \textbf{id}$			

MATCHED	STACK	INPUT	Action
	$E\$$	$\textbf{id} + \textbf{id} * \textbf{id}\$$	
	$TE'\$$	$\textbf{id} + \textbf{id} * \textbf{id}\$$	output $E \rightarrow TE'$
	$FT'E'\$$	$\textbf{id} + \textbf{id} * \textbf{id}\$$	output $T \rightarrow FT'$
	$\textbf{id}\ T'E'\$$	$\textbf{id} + \textbf{id} * \textbf{id}\$$	output $F \rightarrow \textbf{id}$
$\textbf{id}$	$T'E'\$$	$+ \textbf{id} * \textbf{id}\$$	match $\textbf{id}$
$\textbf{id}$	$E'\$$	$+ \textbf{id} * \textbf{id}\$$	output $T' \rightarrow \epsilon$
$\textbf{id}$	$+TE'\$$	$+ \textbf{id} * \textbf{id}\$$	output $E' \rightarrow +TE'$
$\textbf{id} +$	$TE'\$$	$\textbf{id} * \textbf{id}\$$	match $+$
$\textbf{id} +$	$FT'E'\$$	$\textbf{id} * \textbf{id}\$$	output $T \rightarrow FT'$
$\textbf{id} +$	$\textbf{id}\ T'E'\$$	$\textbf{id} * \textbf{id}\$$	output $F \rightarrow \textbf{id}$
$\textbf{id} + \textbf{id}$	$T'E'\$$	$* \textbf{id}\$$	match $\textbf{id}$
$\textbf{id} + \textbf{id}$	$*FT'E'\$$	$* \textbf{id}\$$	output $T' \rightarrow *FT'$
$\textbf{id} + \textbf{id} *$	$FT'E'\$$	$\textbf{id}\$$	match $*$
$\textbf{id} + \textbf{id} *$	$\textbf{id}\ T'E'\$$	$\textbf{id}\$$	output $F \rightarrow \textbf{id}$
$\textbf{id} + \textbf{id} * \textbf{id}$	$T'E'\$$	$\$$	match $\textbf{id}$
$\textbf{id} + \textbf{id} * \textbf{id}$	$E'\$$	$\$$	output $T' \rightarrow \epsilon$
$\textbf{id} + \textbf{id} * \textbf{id}$	$\$$	$\$$	output $E' \rightarrow \epsilon$

Panic Mode Recovery

Add synchronizing actions to undefined entries based on FOLLOW

FOLLOW(E) = { **) $** }
FOLLOW(E') = { **) $** }
FOLLOW(T) = { **+) $** }
FOLLOW(T') = { **+) $** }
FOLLOW(F) = { **+ *) $** }

Example: As $ ∈ Follow(E), M(E, $) = ***synch***

	id	**+**	*****	**(**	**)**	**$**
E	$E \to T E'$			$E \to T E'$	***synch***	***synch***
E'		$E' \to + T E'$			$E' \to \varepsilon$	$E' \to \varepsilon$
T	$T \to F T'$	***synch***		$T \to F T'$	***synch***	***synch***
T'		$T' \to \varepsilon$	$T' \to * F T'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to$ **id**	***synch***	***synch***	$F \to (E)$	***synch***	***synch***

The driver pops current nonterminal A if M(A, a) in the above table is ***synch***, or skips input (lookahead) token a if M(A, a) is blank.

Example: Moves of parsing and error recovery on the erroneous input **+ *id* * + *id***

STACK	INPUT	REMARK
$E\ \$$	$+\ \mathbf{id} * + \mathbf{id}\ \$$	error, skip $+$
$E\ \$$	$\mathbf{id} * + \mathbf{id}\ \$$	**id** is in FIRST(E)
$TE'\ \$$	$\mathbf{id} * + \mathbf{id}\ \$$	
$FT'E'\ \$$	$\mathbf{id} * + \mathbf{id}\ \$$	
$\mathbf{id}\ T'E'\$$	$\mathbf{id} * + \mathbf{id}\ \$$	
$T'E'\ \$$	$* + \mathbf{id}\ \$$	
$* FT'E'\ \$$	$* + \mathbf{id}\ \$$	
$FT'E'\ \$$	$+ \mathbf{id}\ \$$	error, $M[F, +] = \text{synch}$
$T'E'\ \$$	$+ \mathbf{id}\ \$$	F has been popped
$E'\ \$$	$+ \mathbf{id}\ \$$	
$+ TE'\ \$$	$+ \mathbf{id}\ \$$	
$TE'\ \$$	$\mathbf{id}\ \$$	
$FT'E'\ \$$	$\mathbf{id}\ \$$	
$\mathbf{id}\ T'E'\ \$$	$\mathbf{id}\ \$$	
$T'E'\ \$$	$\$$	
$E'\ \$$	$\$$	
$\$$	$\$$	

	id	**+**	*****	**$**
E	$E \rightarrow T E'$			***synch***
E'		$E' \rightarrow + T E'$		$E' \rightarrow \varepsilon$
T	$T \rightarrow F T'$	***synch***		***synch***
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow * F T'$	$T' \rightarrow \varepsilon$
F	$F \rightarrow \mathbf{id}$	***synch***	***synch***	***synch***

Phrase-Level Recovery

$E \rightarrow T\, E'$
$E' \rightarrow +\, T\, E' \mid \varepsilon$
$T \rightarrow F\, T'$
$T' \rightarrow *\, F\, T' \mid \varepsilon$
$F \rightarrow (\, E\,) \mid \textbf{id}$

Change input stream by inserting missing tokens
For example: **id id** is changed into **id * id**

Pro: Can be fully automated
Cons: Recovery not always intuitive

Can then continue here

	id	+	*	(	)	$
E	$E \rightarrow T\, E'$			$E \rightarrow T\, E'$	***synch***	***synch***
E'		$E' \rightarrow +\, T\, E'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow F\, T'$	***synch***		$T \rightarrow F\, T'$	***synch***	***synch***
T'	***insert ****	$T' \rightarrow \varepsilon$	$T' \rightarrow *\, F\, T'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow$ **id**	***synch***	***synch***	$F \rightarrow (\, E\,)$	***synch***	***synch***

insert *: driver inserts missing * and retries the production

5. Bottom-Up Parsing

- LR methods (Left-to-right, Rightmost derivation)
  - SLR, Canonical LR, LALR
- Other special cases:
  - Shift-reduce parsing
  - Operator-precedence parsing

Shift-Reduce Parsing

Grammar:

$S \rightarrow \mathbf{a}\ A\ B\ \mathbf{e}$

$A \rightarrow A\ \mathbf{b}\ \mathbf{c} \mid \mathbf{b}$

$B \rightarrow \mathbf{d}$

Reducing a sentence:

$\mathbf{a}\ \underline{\mathbf{b}}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ \underline{A\ \mathbf{b}\ \mathbf{c}}\ \mathbf{d}\ \mathbf{e}$

$\mathbf{a}\ A\ \underline{\mathbf{d}}\ \mathbf{e}$

$\underline{\mathbf{a}\ A\ B\ \mathbf{e}}$

S

These match production's right-hand sides

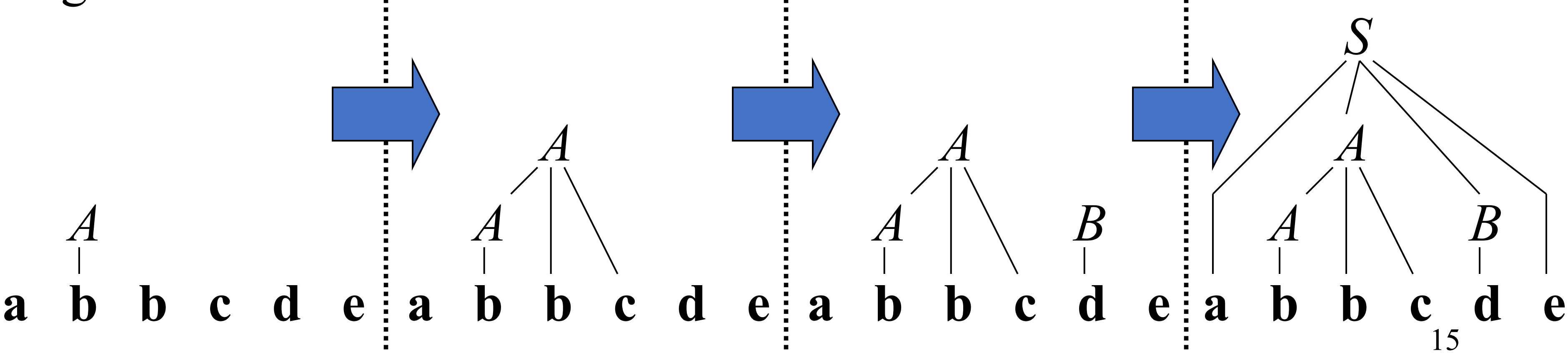
Shift-reduce corresponds to the reverse of a rightmost derivation:

$S \Rightarrow_{rm} \mathbf{a}\ A\ B\ \mathbf{e}$

$\Rightarrow_{rm} \mathbf{a}\ A\ \mathbf{d}\ \mathbf{e}$

$\Rightarrow_{rm} \mathbf{a}\ A\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$

$\Rightarrow_{rm} \mathbf{a}\ \mathbf{b}\ \mathbf{b}\ \mathbf{c}\ \mathbf{d}\ \mathbf{e}$



Handles

A handle is a substring that matches the body of a production, and whose reduction represents one step along the reverse of a rightmost derivation

Grammar:

$S \rightarrow$ **a** A B **e**

$A \rightarrow A$ **b c** | **b**

$B \rightarrow$ **d**

a b b c d e

a A **b c d e**

a A **d e**

a A B **e**

S

Handle

a b b c d e

a A **b c d e**

a A A **e**

… ?

NOT a handle, because further reductions will fail (result is not a sentential form)