COMPILER LECTURES

COMPUTER SCIENCE

3RD CLASS

M.SC. SAMER AL-YASSIN

2017-2018

LECTURE 7

Bottom-Up Parsing

Is attempting to construct a parse tree for an input string beginning from leaves (the bottom) and working up towards the root (the top). In this section, there is a general style of bottom-up syntax analysis, known as ***Shift-Reduce Parsing***. An easy to implement form of this parsing, called ***Operator-Precedence Parsing*** a much more general method of Shift-Reduce Parsing, called ***LR Paring***, used in a number of automatic parser generators.

In this lecture, we introduce a general style of bottom-up syntax analysis, known as ***shift-reduce parsing***. An easy-to-implement form of shift-reduces parsing, called ***operator-precedence parsing***, is presented in next sections. A much more general method of shift-reduce parsing, called ***LR parsing***. LR parsing is used in a number of automatic parser generators.

Shift-Reduce Parsing:

Shift-reduce parsing attempts to construct a parse tree for an input string beginning at the leaves (the bottom) and working up towards the root (the top). We can think of this process as one of "**reducing**" a string ***w*** to the start symbol of a grammar. At each **reduction** step a particular substring matching the ***right side of a production*** is replaced by the symbol on the ***left*** of that production, and if the substring is chosen correctly at each step, a ***rightmost derivation*** is traced out in reverse.

Example: Consider the grammar:

$S \longrightarrow aABe$
$A \longrightarrow Abc \mid b$
$B \longrightarrow d$

The sentence ***abbcde*** can be reduced to S by the following steps:

abbcde
aAbcde
aAde
aABe
S

We scan ***abbcde*** looking for a substring that matches the right side of some production. These reductions, in fact, trace out the following **rightmost** derivation in reverse:

$$abbcde \xrightarrow{rm} aAbcde \xrightarrow{rm} aAde \xrightarrow{rm} aABe \xrightarrow{rm} S$$

- **Handles**

Informally, a "**handle**" of a string is a substring that matches the right side of a production, and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation. In many cases the leftmost substring **β** that matches the right side of some production $A \rightarrow \beta$ is not a **handle**, because a reduction by the production $A \rightarrow \beta$ yields a string that cannot be reduced to the **start symbol**.

In previous example, if we replaced **b** by ***A*** in the second string ***aAbcde*** we would obtain the string ***aAAcde*** that cannot be subsequently reduced to ***S***. For this reason, we must give a more precise definition of a handle.

Formally, a ***handle*** of a right-sentential form **γ** is a production ***A*** → **β** and a position of **γ** where the string **β** may be found and replaced by ***A*** to produce the previous right-sentential form in a rightmost derivation of **γ**.

- **Stack Implementation of Shift-Reduce Parsing**

There are two problems that must be solved if we are to parse by handle. The first is to locate the substring to be reduced in a right-sentential form, and the second is to determine what production to choose in case there is more than one production with that substring on the right side. Before we get to these questions, let us first consider the type of data structures to use in a shift-reduce parser.

A convenient way to implement a **Shift-Reduce Parsing** is to use **stack** to hold grammar symbols and an **input buffer** to hold the string (***W***) to be parsed. ***Use $ to mark the bottom of the stack and also the right end of the input.***

Initially, the stack is empty, and the string (***W***) is on the input as follows:

Stack	**Input**
$	***W***$

The parser operates by shifting zero or more input symbols onto the stack until a handle **β** is on top of the stack. The parser then reduces **β** to left side of the appropriate production. The parser repeat this cycle until it has detected an error or until the stack contains the start symbol ***S*** and the input is empty:

Stack	**Input**
$**S**	$

After entering this configuration, the parser halts and announces successful completion of parsing.

Note: the operations of the parser (**Shift**, **Reduce**, **Accept**, and **Error "not accept"**).

Example: Parse the input **id1+id2*id3** for this grammar:

$$E \longrightarrow E+E \;/\; E*E \;/\; (E) \;/\; id$$

Stack	Input	Action
$	id1+id2*id3$	Shift
$id1	+id2*id3$	Reduce E ⟶ id
$E	+id2*id3$	Shift
$E+	id2*id3$	Shift
$E+id2	*id3$	Reduce E ⟶ id
$E+E	*id3$	Shift
$E+E*	id3$	Shift
$E+E*id3	$	Reduce E ⟶ id
$E+E*E	$	Reduce E ⟶ E*E
$E+E	$	Reduce E ⟶ E+E
$E	$	Accept

Note: There is another sequence of steps a shift-reduce parser could take because the grammar is **ambiguous**.

Example: parse the input **id +*id** for same grammar above:

Stack	Input	Action
$	id +*id	Shift
$id	+ *id$	Reduce E ⟶ id
$E+	*id$	Shift
$E+*	id$	Shift
$E+*id	$	Shift
$E+*E	$	Reduce E ⟶ id
$E+*E	$	Error

- **Operator-Precedence Parsing (OPP)**

The largest class of grammars for which shift-reduce parsers can be built successfully. However, for a small but important class of grammars we can easily construct efficient shift-reduce parsers by hand. These grammars have the **property** (among other essential requirements) that **no production right side is Є or** has **two adjacent nonterminals**. A grammar with the latter property is called an **operator grammar**.

Example: The following grammar for expressions

E ⟶ EAE / (E) / -E / id

A ⟶ + /- / * / ÷ / ↑

Is not an **operator grammar**, because the right side **EAE** has two (in fact three) consecutive nonterminals. However, if we substitute for **A** each of its alternatives, we obtain the following operator grammar:

E ⟶ E+E / E-E / E*E / E÷E / E↑E / (E) / -E / id

We now describe an easy-to-implement parsing technique called operator-precedence parsing.

In operator-precedence parsing, we define three disjoint **precedence relations**, **<•, =, and •>**, between certain pairs of **terminals**. These precedence relations guide the selection of **handles** and have the following meanings:

RELATION	MEANING
a* <• *b	***a*** *"yields precedence to"* ***b***
a* = *b	***a*** *"has the same precedence as"* ***b***
a* •> *b	***a*** *"takes precedence over"* ***b***

Example: The following grammar for expressions

E ⟶ E+E / E*E / id

For example, suppose we initially have the right-sentential form **id + id * id** and the precedence relations are shown in the table below.

	id	+	*	$
id		•>	•>	•>
+	<•	•>	<•	•>
*	<•	•>	•>	•>
$	<•	<•	<•	

Operator-precedence relations

Then the string with the precedence relations inserted is:

$ <• id •> + <• id •>* <• id •>$
$ E + <• id •> * <•id •> $
$ <•E + <• id •> * <•id •> $
$ <•E + E * <•id •> $
$ <•E + <• E * <•id •> $
$ <•E + <• E * E $
$ <•E + <• E * E•> $
$ <•E + E $
$ <•E + E•>$
$ E$ Accept

<u>Table Construction of Operator-Precedence Relations</u>

The table of Operator-precedence relations can be created according to the following steps:

1- Compute the **LEADING** and **TRAILING** for each nonterminal.
2- Determine the relation for each two terminal symbols ***a*** and ***b***.

LEADING & TRAILING

LEADING(***A***) ={a | $A \xrightarrow{+} \gamma a \delta$, where **γ** is **Є** or single nonterminal}
TRAILING(***A***)={a | $A \xrightarrow{+} \gamma a \delta$,where **δ** is **Є** or single nonterminal}

Example: The following grammar for expressions

$E \longrightarrow E+T \mid T$
$T \longrightarrow T*F \mid F$
$F \longrightarrow (E) \mid id$

Nonterminals	LEADING	TRAILING
E	**+,*, (, id**	**+, *,), id**
T	***, (, id**	***,), id**
F	**(, id**	**), id**

- **Relations of Operator-Precedence Table**

For each two terminal symbols **a** and **b**, we say:

1) **a = b** if there is a right side of a production of the form **αaβbγ**, where **β** is either **Є** or a single nonterminal. That is **a = b** if **a** appears immediately to the left of **b** in a right side, or if they appear separated by one nonterminal. For example, the production $S \longrightarrow iCtSeS$ implies that **i = t and t = e**.
2) **a <• b** if for some nonterminal ***A*** there is a right side of the form **αa*A*β**, then **a <• LEADING (*A*)**

   For Example, $S \longrightarrow iCtS$, and $C \xrightarrow{+} b$, so **i <• b**. and **t <• i**

   Also, the **$ <• LEADING (*S*)**, **where S is start Symbol.**

3) **a •> b** if for some nonterminal ***A*** there is a right side of the form **α*Ab*β**, then **TRAILING(*A*) •> b**

   For Example, $S \longrightarrow iCtS$, and $C \xrightarrow{+} b$, so **b •> t**.

   Also, the **TRAILING (*A*) •> $**, **where S is start Symbol.**

Example: The following grammar for expressions

$E \longrightarrow E+T \mid T$

$T \longrightarrow T*F \mid F$

$F \longrightarrow (E) \mid \text{id}$

Nonterminals	LEADING	TRAILING
E	+,*, (, id	+, *,), id
T	*, (, id	*,), id
F	(, id	), id

By applying the **Relations** in above will be resulted the following table:

1- a = b, $A a \beta b \gamma$ $(E) \Rightarrow (=)$

2- a <• b, $\alpha a A \beta$

$E+T \Rightarrow$ + <• LEADING(*T*) + <• {*, (, id}

$T*F \Rightarrow$ * <• LEADING(*F*) * <• { (, id}

(E) $\Rightarrow$ (<• LEADING(*E*) (<• {+,*, (, id}

$ <• LEADING (*E*) $ <• {+,*, (, id}

3- a •> b, $\alpha A b \beta$ $E+T \Rightarrow$ TRAILING (E) •> + {+, *,), id} •> +

$T*F \Rightarrow$ TRAILING (T) •> * {*,), id} •> *

$(E) \Rightarrow$ TRAILING (E) •>) {+, *,), id} •>)

TRAILING (E) •> $ {+, *,), id} •> $

	+	*	(	)	id	$
+	•>	<•	<•	•>	<•	•>
*	•>	•>	<•	•>	<•	•>
(	<•	<•	<•	=	<•	
)	•>	•>		•>		•>
id	•>	•>		•>		•>
$	<•	<•	<•		<•	