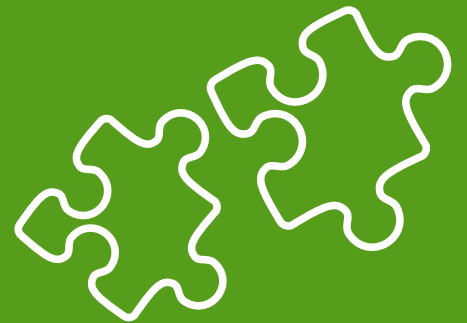


# Lecture 8

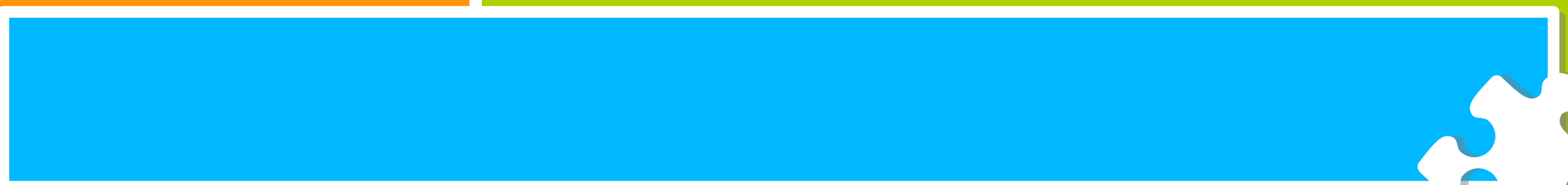


# Bottom up parsing

## “Shift-Reduce” Parsing

**Reduce a string to the start symbol of the grammar.**

At every step a particular **sub-string is matched** (in left-to-right fashion) to the right side of some production and then it is substituted by the non-terminal in the left hand side of the production.



$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

Rightmost Derivation:

$S \Rightarrow aABe \Rightarrow aAde \Rightarrow aAbcde \Rightarrow abbcde$

abbcde

aAbcde

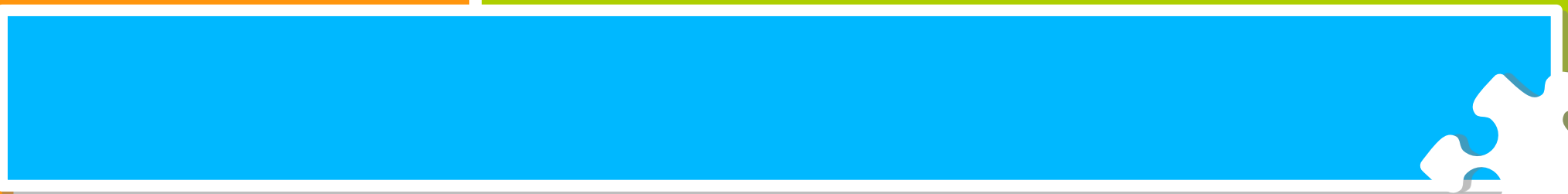
aAde

aABe

S

# Handles

**Substring that matches the RHS of some production AND** whose reduction to the non-terminal on the LHS is a step along the reverse of some rightmost derivation.



$S \rightarrow aABe$

$A \rightarrow Abc \mid b$

$B \rightarrow d$

$a\underline{b}bcde \rightarrow a\underline{A}bcde \rightarrow aA\underline{d}e \rightarrow \underline{aABe} \rightarrow S$

# Shift Reduce Parsing with a Stack



Two problems

- locate a handle
- decide which production to use

Using stack

- **Shift** -- input symbols into the stack until a handle is found on top of it.
- **Reduce** -- the handle to the corresponding non-terminal.
- **Accept** -- when the input is consumed and only the start symbol is on the stack,
- **Error** -- when it does not lead to the start symbol

$E \rightarrow E + E \mid E * E \mid ( E ) \mid id$

Reduce the string  $id + id * id$  using the above grammar.

STACK	INPUT	Remark
\$	id + id * id\$	Shift
\$ id	+ id * id\$	Reduce by $E \rightarrow id$
\$E	+ id * id\$	Shift
\$E+	id * id\$	Shift
\$E+id	* id \$	Reduce $E \rightarrow id$
\$E + E	* id \$	Reduce $E \rightarrow E + E$
\$E	id\$	Shift
\$E *	id\$	Shift
\$E * id	\$	Reduce $E \rightarrow id$
\$E * E	\$	Reduce $E \rightarrow E * E$
\$E	\$	Accept
\$		

## Operator Grammar



It is a bottom-up parser **none of whose productions contain two or more consecutive NTs in any RHS alternative**

**Non terminals occurring in RHS string are separated by one or more terminal symbols**

All terminal symbols occurring in RHS strings are called operators of the grammar





Not an operator grammar

$$E \rightarrow E \text{ op } E \mid \text{id}$$
$$\text{op} \rightarrow + \mid *$$

An operator grammar

$$E \rightarrow E + E \mid E * E \mid \text{id}$$

## Operator precedence table

Operator precedence parser can parse some of the ambiguous grammar by constructing an operator precedence (relations) table

$E \rightarrow E + E \mid E * E \mid id$

	id	+	*	\$
id		$\cdot >$	$\cdot >$	$\cdot >$
+	$< \cdot$	$\cdot >$	$< \cdot$	$\cdot >$
*	$< \cdot$	$\cdot >$	$\cdot >$	$\cdot >$
\$	$< \cdot$	$< \cdot$	$< \cdot$	

# Example

We parse the string

a)  $\text{id} + \text{id} * \text{id}$

b)  $\text{id} + \text{id} + \text{id}$

Using the grammar before and following operator precedence table

$$E \rightarrow E + E \mid E * E \mid id$$

$$id + id * id$$

Stack	Input	Action
\$	id + id * id \$	\$ < . id, shift
\$id	+ id * id \$	id . > +, reduce $E \rightarrow id$
\$	+ id * id \$	\$ < . id, shift
\$+	id * id \$	+ < . id, shift
\$+id	* id \$	id . > +, reduce $E \rightarrow id$
\$+	* id \$	+ < . *, shift
\$+*	id \$	* < . id, shift
\$+*id	\$	id . > \$, reduce $E \rightarrow id$
\$+*	\$	* . > \$, reduce $E \rightarrow E * E$
\$+	\$	+ . > \$, reduce $E \rightarrow E + E$
\$	\$	Accept

	id	+	*	\$
id		.>	.>	.>
+	<.	.>	<.	.>
*	<.	.>	.>	.>
\$	<.	<.	<.	

Note: if  $op1 < op2$ ,  
shift operation  
Otherwise reduce  
operation

## Just to Note...



( =. )

\$ <. (

id >.)

) > \$

( <. (

\$ <. id

id > \$

) > )

( <. id

The rules for operator precedence will also be based on the precedence of operators

## Exercise

Create an operator precedence table for the grammar

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid E ^ E \mid \text{id}$$

Using the operator precedence table, reduce the string

$$\text{id} * \text{id} ^ \text{id} + \text{id}$$

# Disadvantages



## **Cannot handle unary minus**

Solution: let lexical analyser handle

1. The lexical analyser will return two different tokens for the unary minus and the binary minus
2. The lexical analyser will need a lookahead to distinguish the binary minus from the unary minus



Hence, we make

- $\theta < \cdot$  unary-minus  $\rightarrow$  *for any operator*
- unary-minus  $\cdot > \theta$   $\rightarrow$  *if unary-minus has higher precedence than  $\theta$*
- unary-minus  $< \cdot \theta$   $\rightarrow$  *if unary-minus has lower (or equal) precedence than  $\theta$*



# Advantages



- Simple
- Powerful enough for expressions in programming languages

# LALR parser

Kindly self study this topic

