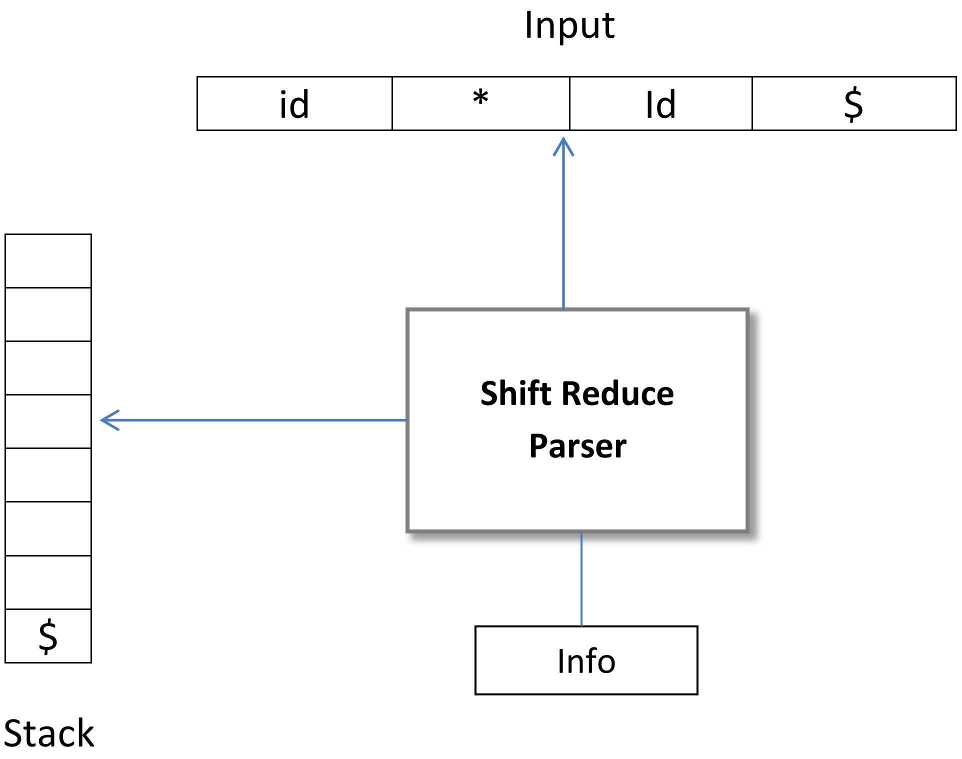
SHIFT REDUCE PARSER

September 02, 2013, Report
Saurav Manchanda
(11CS10040)



Consider following productions:

E →E+T|T
T →T*F|F
F → id

Stack	String	Acion
$	Id*id$	Shift
$id	*id$	Reduce (F → id)
$F	*id$	Reduce (T →F)
$T	*id$	Shift
$T*	id$	Shift
$T*id	$	Reduce (F → id)

$T*F	$	Reduce (T →T*F)
$T	$	Reduce (E →T)
$E	$	Accept

Eventually,
Stack: $S String: $ → Accept State

Challenges:

1. Stack top will always match with the rightmost symbol. Real challenge is to find the leftmost symbol in stack.
2. Shift-Reduce Conflict: Decision whether to shift or reduce.
3. Reduce-Reduce Conflict: Which production to use for reduction out of following:

B→γ
C→γ

Different Parsers:

1. Operator Precedence Parser
2. LR Parser
    i) SLR
    ii) CLR
    iii) LALR

Top Down parsers works for only a subset of CFG, i.e... LL(1) grammars. Similarly,

- Operator Precedence Parser: Operator Grammar
- LR Parser: LR Grammar

Operator Precedence Parser

- ➢ Arithmetic Expressions.
- ➢ Very difficult to make it work for other grammars.

Operator Grammar

- ➢ No ε-transitions
- ➢ No two or more consecutive non-terminals.

Operator Precedence Info:

For a,b ϵ T

1) a has higher precedence over b ⇨ a·>b
2) a has equal precedence with b ⇨ a·=b
3) a has lower precedence than b ⇨ a<·b

	id	+	*	$
id	X	·>	·>	·>
+	<·	·>	<·	·>
*	<·	·>	·>	·>
$	<·	<·	<·	X

In case of equal precedence, check the associativity of that particular operator.

id always has highest precedence and $ has lowest precedence.