Introduction to Parsing Using Lex/Flex and Yacc/Bison

Uday Khedker

(www.cse.iitb.ac.in/~uday)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

Jan 2025

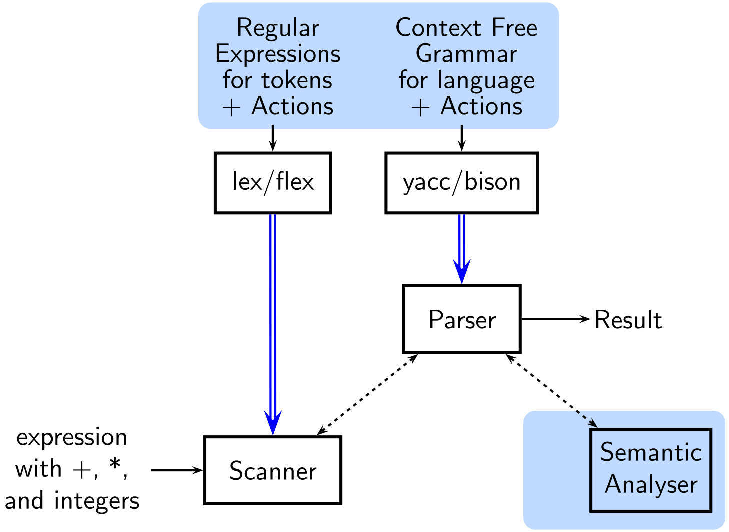
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

simCalc: A Simple Calculator



Regular Expressions for tokens + Actions
Context Free Grammar for language + Actions
lex/flex
yacc/bison
Parser
Result
Scanner
expression with +, *, and integers
Semantic Analyser

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

Introduction to Lex and Yacc

- lex script exp.l to show token identification
- exp1.y to show a simple expression. Use the lex script exp.l
- exp2.y to show construction of a PLUS expression

  Interesting input: `1 + 2 + 3 + 4 + 5 + 6`
- exp3.y fixes the problem by making + left associative
- exp4.y includes + and *

  Interesting inputs: 1 + 2 * 3 and 1 * 2 + 3
- exp5.y fixes the above problem

The Interaction Between Scanner and Parser

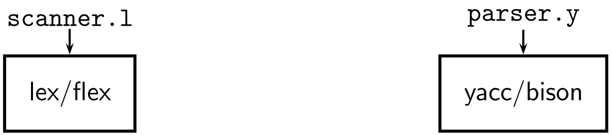
- Grammar

```
Expr  :  Expr '+' Expr
      |  Expr '*' Expr
      |  NUM
```

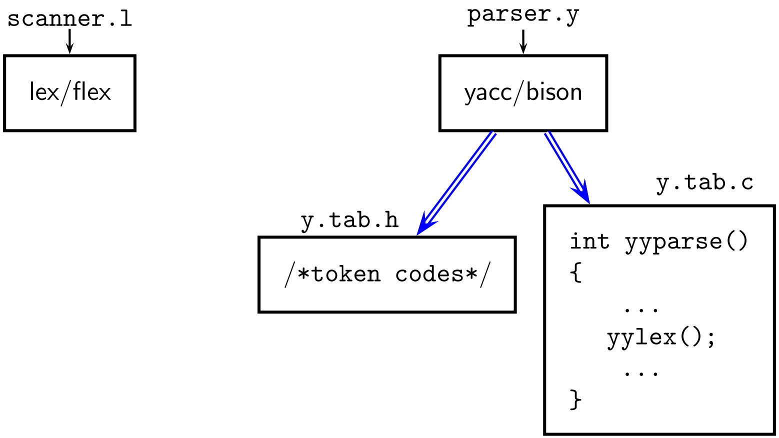
- Terminals and non-terminals get defined by the grammar
- Scanner identifies the tokens and communicates the details to the parser

Token Name	Token Lexemes	Token Value	Token Code
Number	"10"	10	NUM
	"345"	345	
	"03"	3	
+ operator	"+"		'+'
* operator	"*"		'*'

The Interaction Between Scanner and Parser

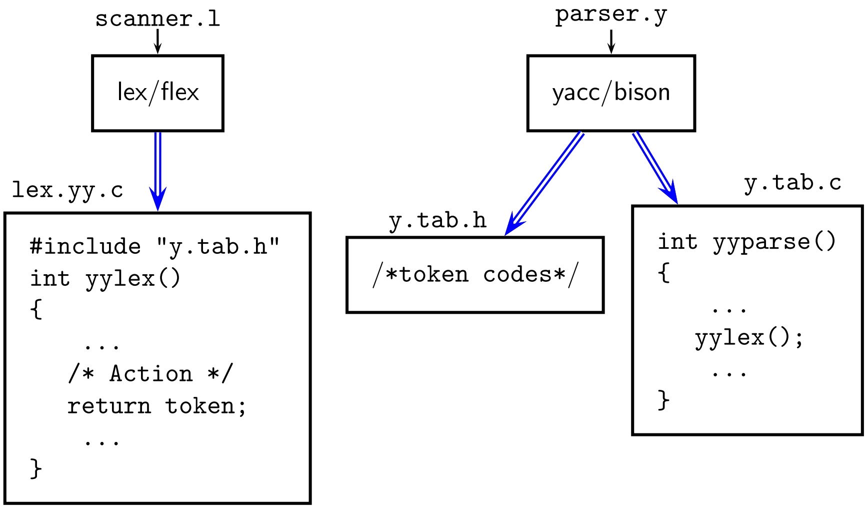


The Interaction Between Scanner and Parser



```
int yyparse()
{
    ...
  yylex();
    ...
}
```

The Interaction Between Scanner and Parser



```
#include "y.tab.h"
int yylex()
{
    ...
   /* Action */
   return token;
    ...
}
```

```
/*token codes*/
```

```
int yyparse()
{
    ...
   yylex();
    ...
}
```

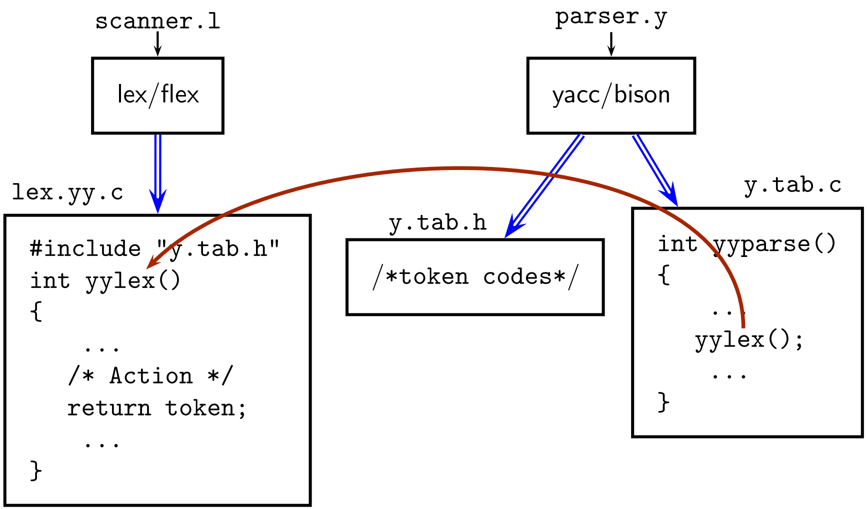
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

The Interaction Between Scanner and Parser

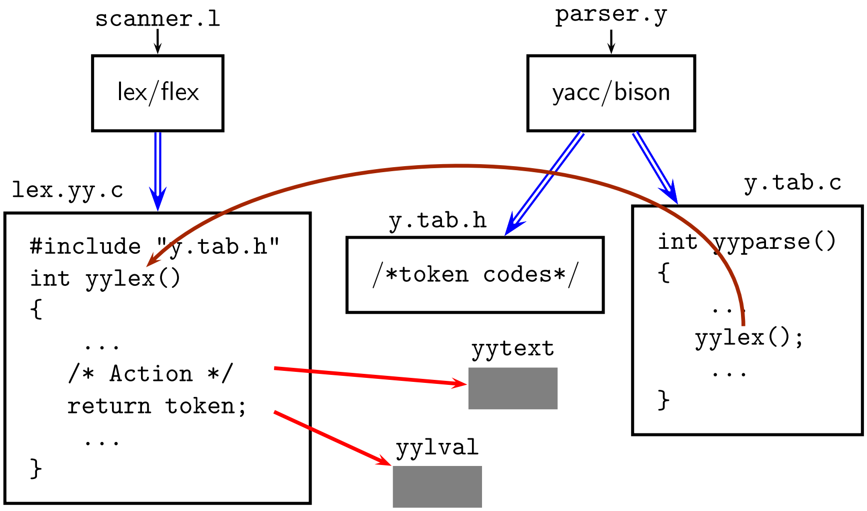


```
#include "y.tab.h"
int yylex()
{
    ...
   /* Action */
   return token;
    ...
}
```

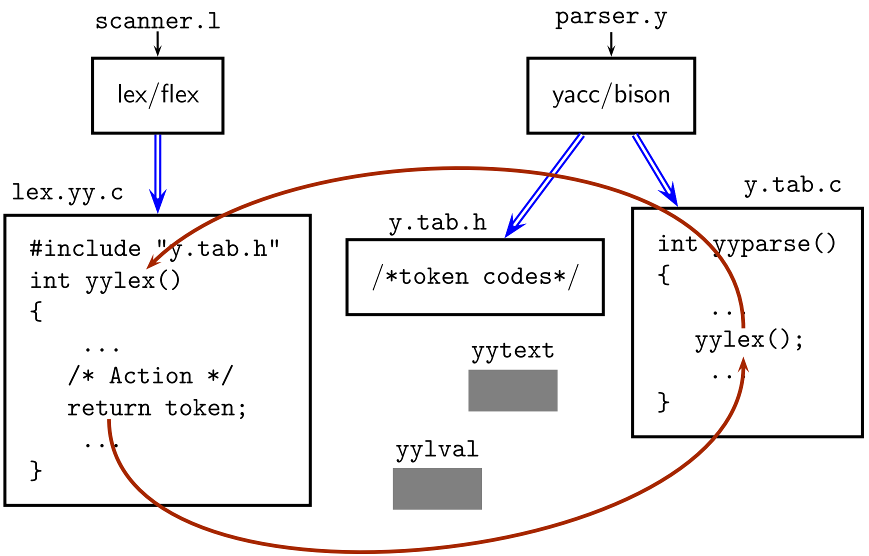
```
/*token codes*/
```

```
int yyparse()
{
     ...
   yylex();
     ...
}
```

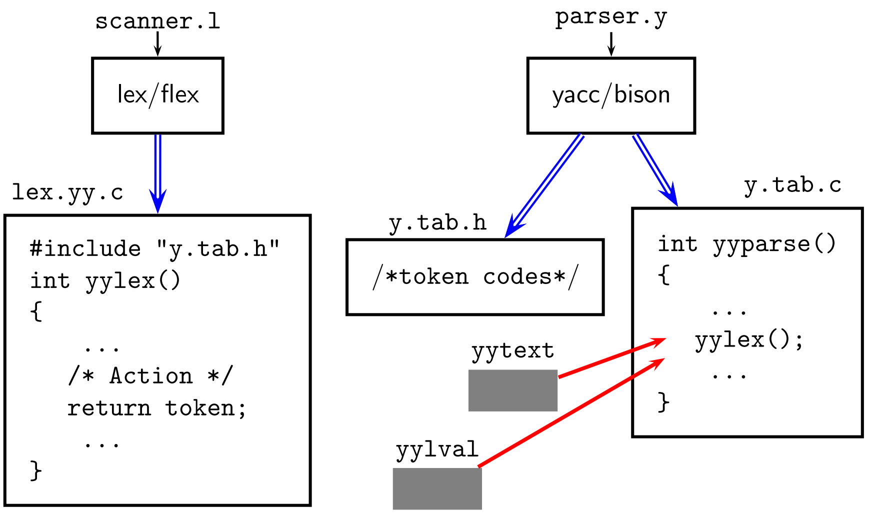
The Interaction Between Scanner and Parser



The Interaction Between Scanner and Parser



The Interaction Between Scanner and Parser



```
#include "y.tab.h"
int yylex()
{
   ...
  /* Action */
  return token;
   ...
}
```

```
/*token codes*/
```

```
int yyparse()
{
    ...
   yylex();
    ...
}
```

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing

$$\begin{array}{cccccc} 10 & + & 20 & * & 3 & \boxed{\text{eof}} \\ \text{NUM} & & & & & \end{array}$$

Parsing Stack

Action: Shift NUM

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

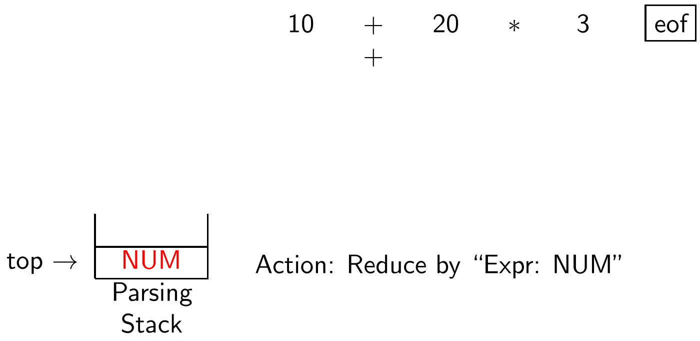
Section:

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



Action: Reduce by “Expr: NUM”

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

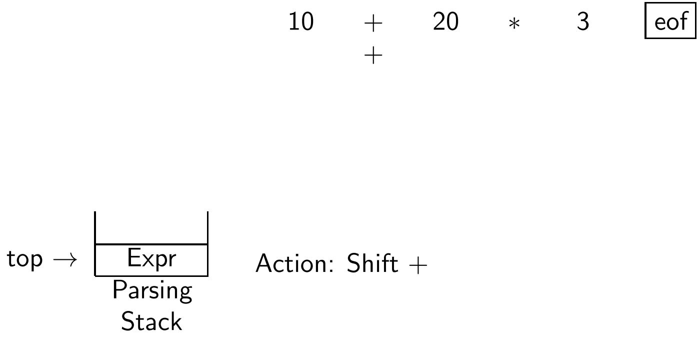
Section:

An Overview of Shift Reduce Parsing

- Grammar

```
Expr  :  Expr '+' Expr
      |  Expr '*' Expr
      |  NUM
```

- The process of parsing



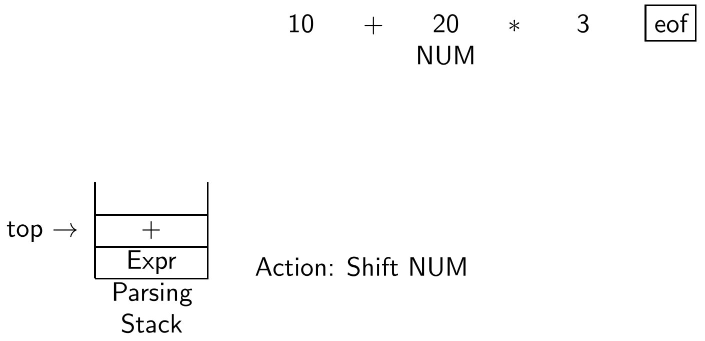
Action: Shift +

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



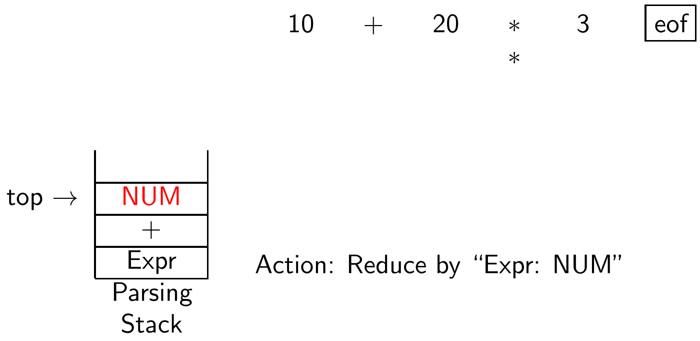
Action: Shift NUM

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



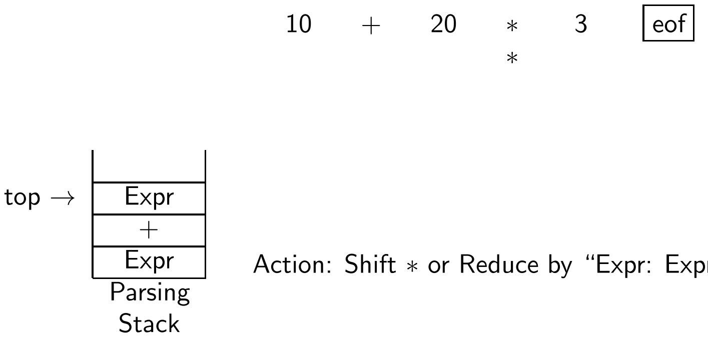
Action: Reduce by "Expr: NUM"

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



Action: Shift ∗ or Reduce by "Expr: Expr + Expr"?

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

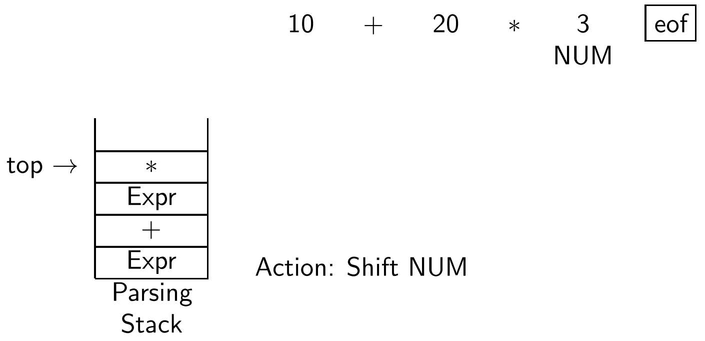
Section:

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



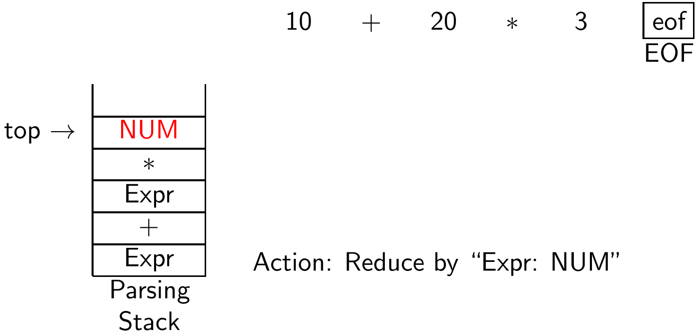
Action: Shift NUM

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



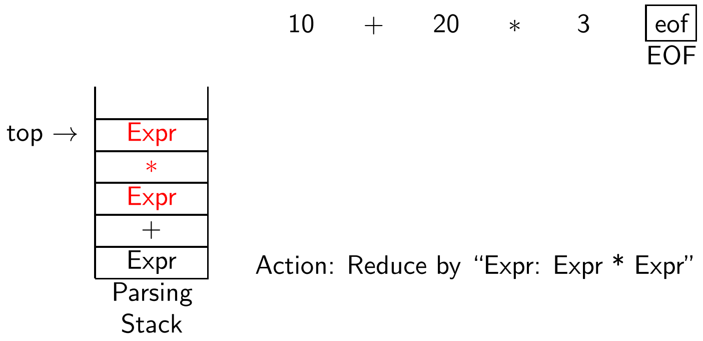
Action: Reduce by "Expr: NUM"

An Overview of Shift Reduce Parsing

- Grammar

```
Expr   :   Expr '+' Expr
       |   Expr '*' Expr
       |   NUM
```

- The process of parsing



Action: Reduce by “Expr: Expr * Expr”

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

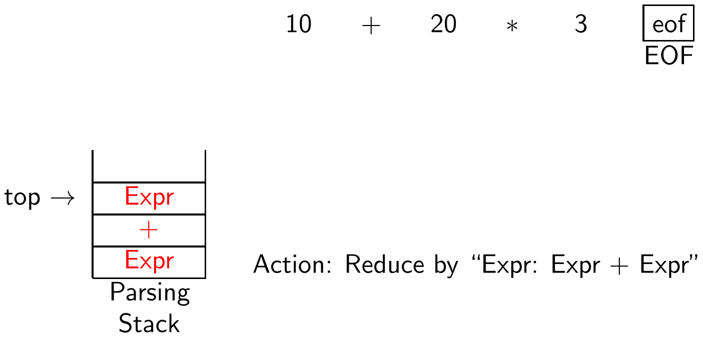
Section:

An Overview of Shift Reduce Parsing

- Grammar

```
Expr  :  Expr '+' Expr
      |  Expr '*' Expr
      |  NUM
```

- The process of parsing



Action: Reduce by “Expr: Expr + Expr”

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

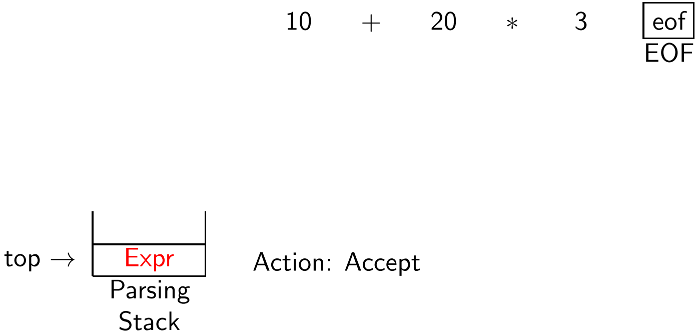
Section:

An Overview of Shift Reduce Parsing

- Grammar

```
Expr  :  Expr '+' Expr
      |  Expr '*' Expr
      |  NUM
```

- The process of parsing



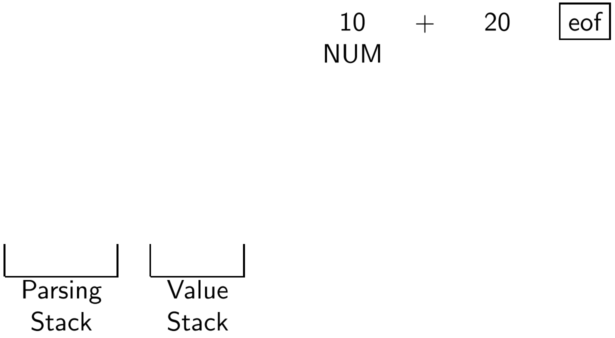
Action: Accept

An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr   :   Expr '+' Expr    { $$ = $1 + $3;}
       |   NUM
```

- The process of parsing and attribute evaluation

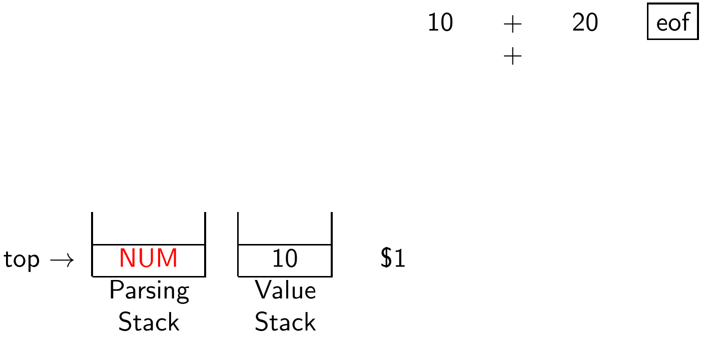


An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr   :   Expr '+' Expr    { $$ = $1 + $3;}
       |   NUM
```

- The process of parsing and attribute evaluation

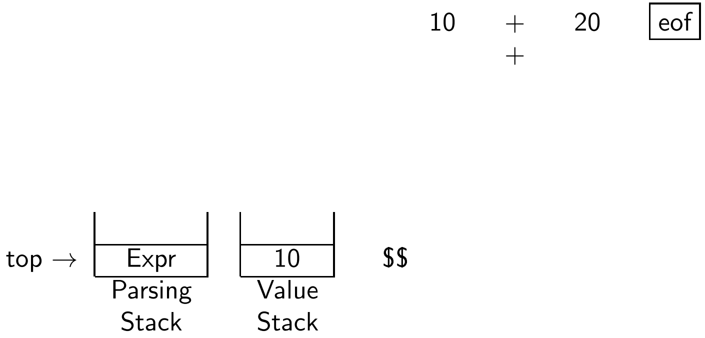


An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr   :   Expr '+' Expr     { $$ = $1 + $3;}
       |   NUM
```

- The process of parsing and attribute evaluation

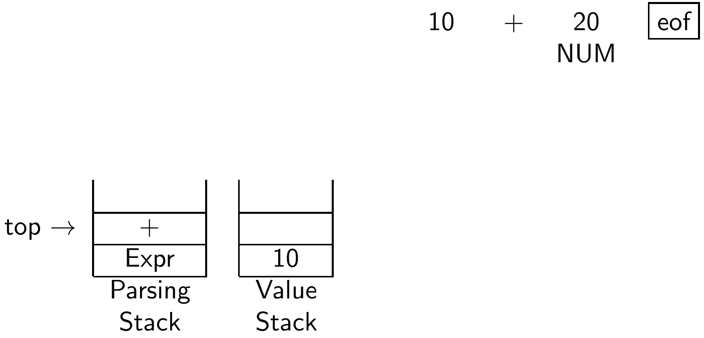


An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr    :   Expr '+' Expr     { $$ = $1 + $3;}
        |   NUM
```

- The process of parsing and attribute evaluation

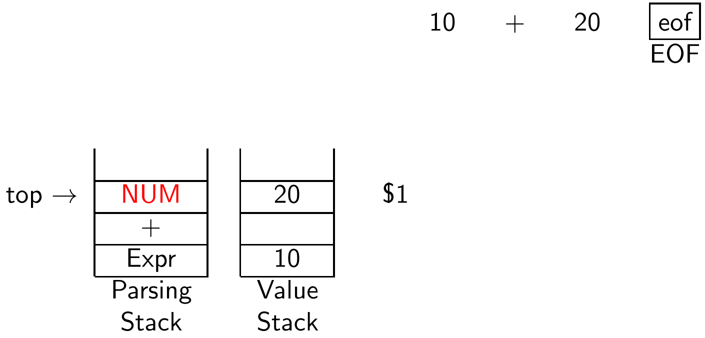


An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr  :  Expr '+' Expr    { $$ = $1 + $3;}
      |  NUM
```

- The process of parsing and attribute evaluation



IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

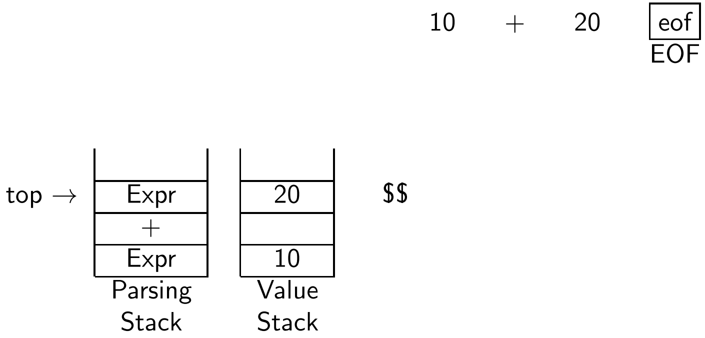
Section:

An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr  :  Expr '+' Expr    { $$ = $1 + $3;}
      |  NUM
```

- The process of parsing and attribute evaluation

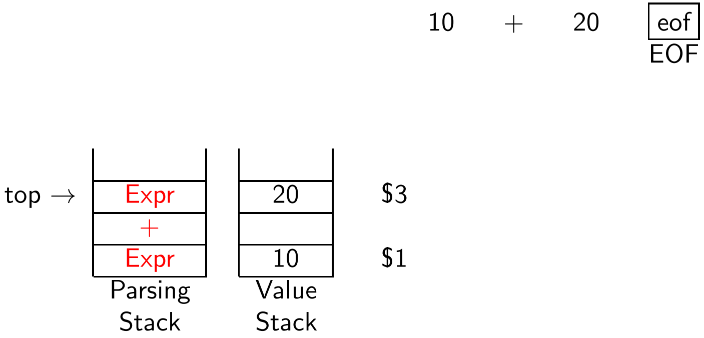


An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr   :   Expr '+' Expr     { $$ = $1 + $3;}
       |   NUM
```

- The process of parsing and attribute evaluation



IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

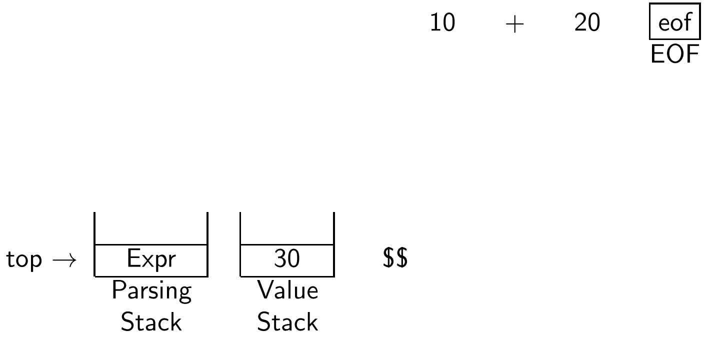
Section:

An Overview of Attribute Evaluation

- Grammar and the associated actions

```
Expr   :   Expr '+' Expr     { $$ = $1 + $3;}
       |   NUM
```

- The process of parsing and attribute evaluation



How to Get Started with Assignment A2 (1)

Study the code of compiler-interpreter-with-TAC-data-structure
Understand the following:

- The role of %union in a yacc script
  - Defines the type of value stack as a C union
  - Type annotations of grammar symbols (using the field names of the union) in the yacc script enable the selection of appropriate field of the union
  - The union becomes available in the lex script by including the .tab.h file
  - Appropriate field name is used with the yylval variable
  - The value of yylval (with appropriate field name) gets pushed on the value stack whenever a token is pushed on the parsing stack
- The Expression_Attributes, Code, TAC_Statement, and TAC_Opd classes and their relationships

IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

How to Get Started with Assignment A2 (2)

- Add a SymbolTable class to store variables and their types

  Create separate tables for global and local variables

  Hint: Check the --show-symtab option of the reference implementation
- Create AST data structure
  - Use class hierarchy and virtual functions wisely
  - Do not store strings representing variables, operators, or numbers in AST nodes

    Use numbers, enums, and pointers (the only exception is a string constant)
- Check types
  - The types of operands should match with the allowable types for each operator

    Hint: Store the type of an expression in the root node of its AST
  - The type of LHS of an assignment should match with that of the RHS expression

Hint: First construct the ASTs without worrying about the types, process the declarations later and then add the type checking code

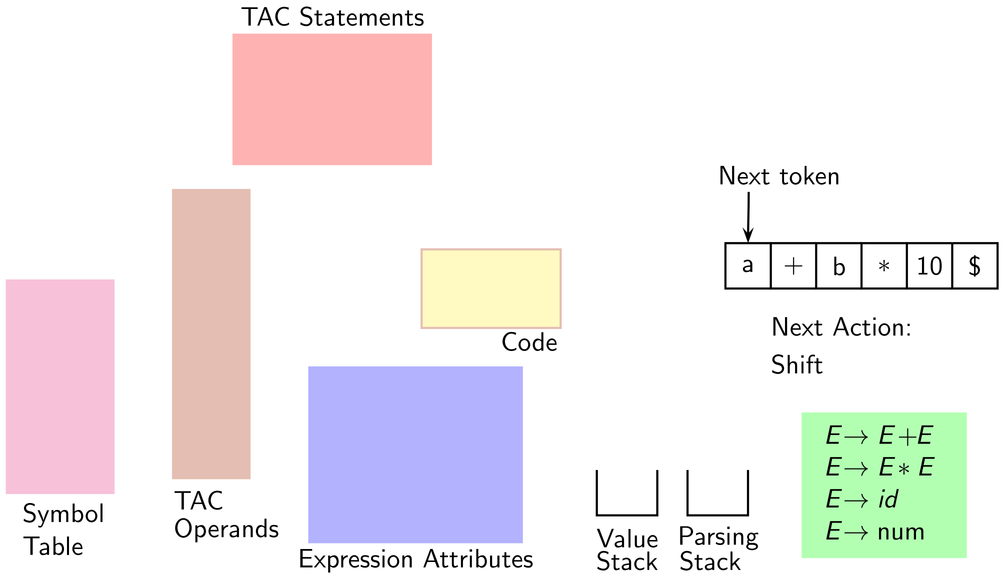
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

Constructing TAC Statements During Parsing

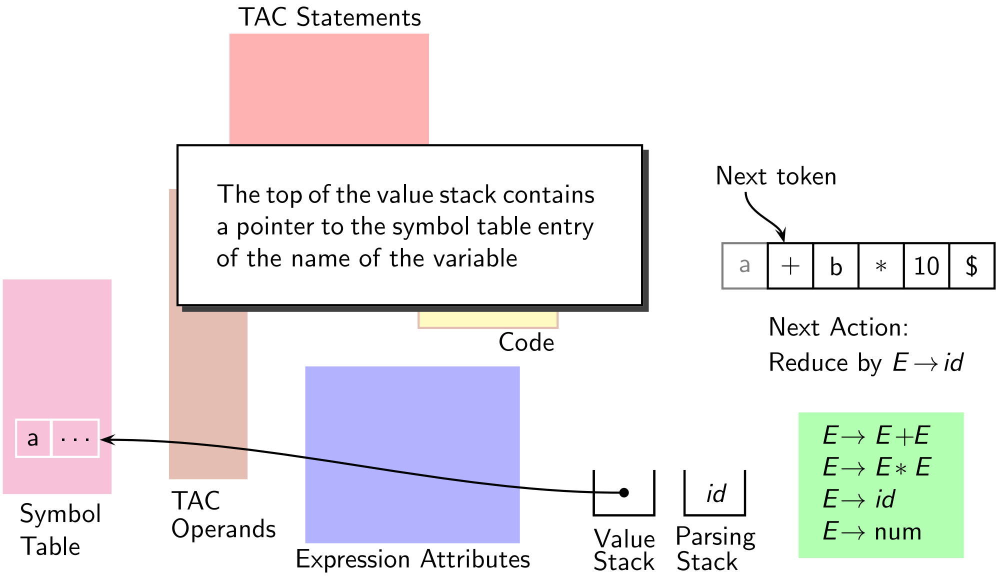


Next Action:

Shift

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

Constructing TAC Statements During Parsing

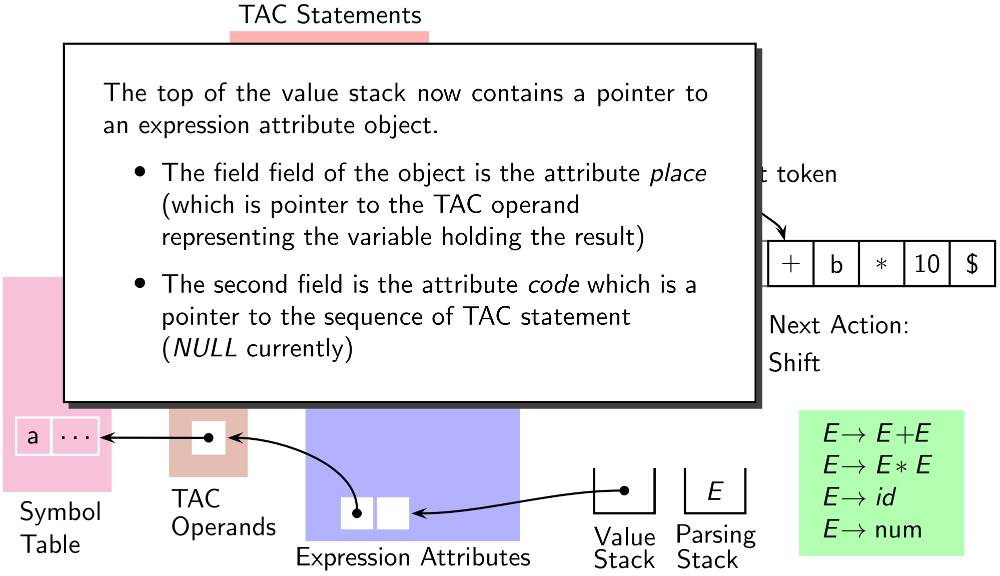


The top of the value stack contains a pointer to the symbol table entry of the name of the variable

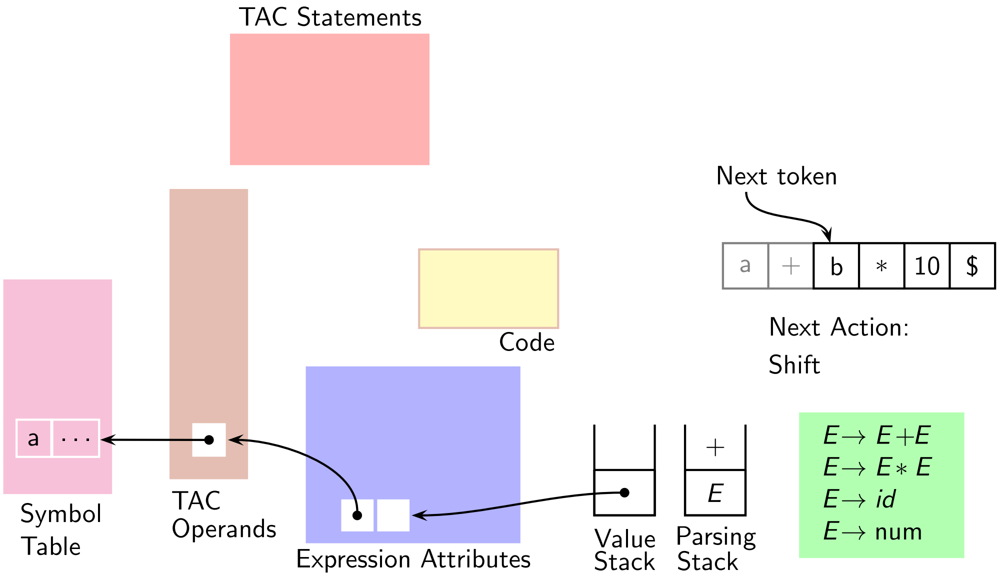
Constructing TAC Statements During Parsing

The top of the value stack now contains a pointer to an expression attribute object.

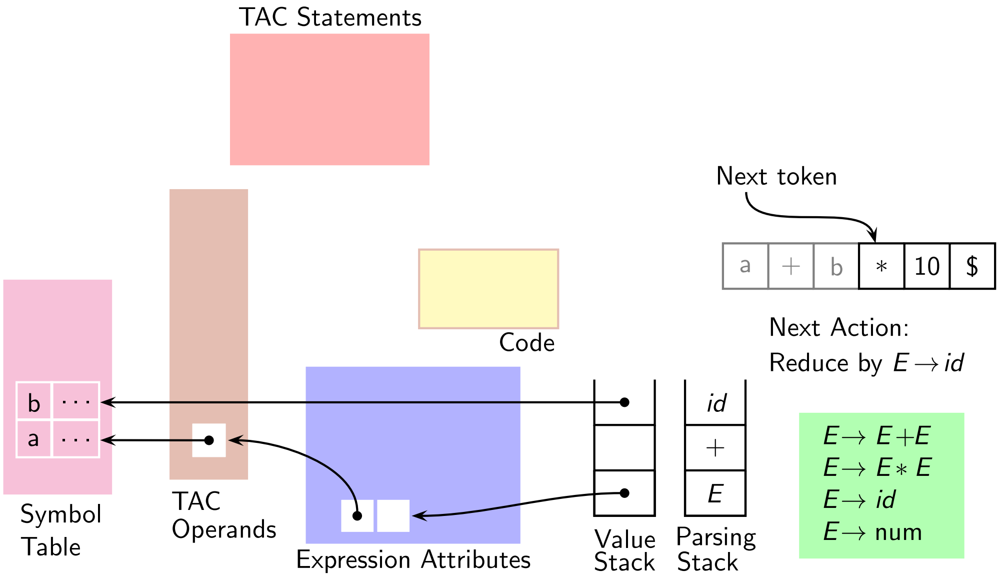
- The field field of the object is the attribute *place* (which is pointer to the TAC operand representing the variable holding the result)
- The second field is the attribute *code* which is a pointer to the sequence of TAC statement (*NULL* currently)



Constructing TAC Statements During Parsing



Constructing TAC Statements During Parsing



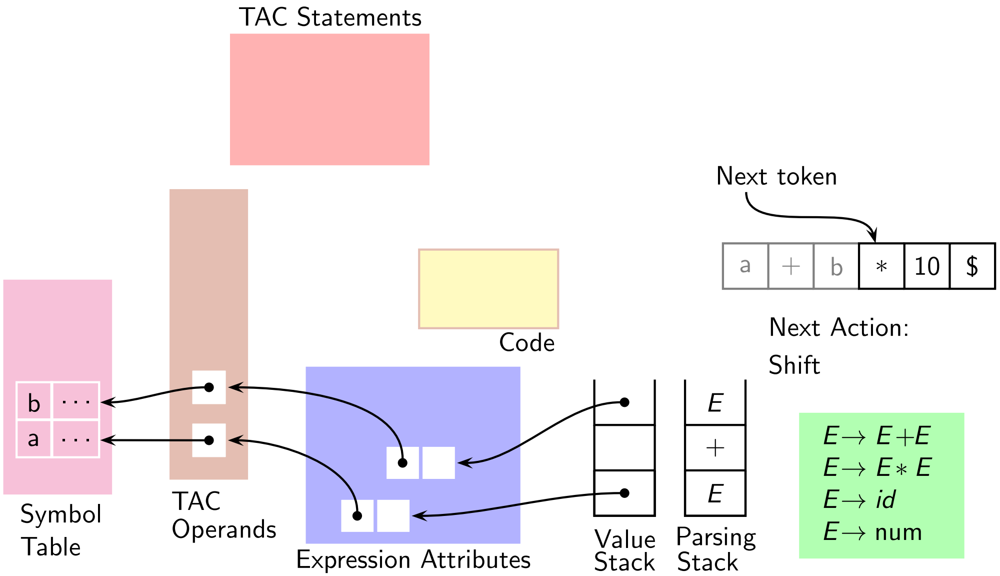
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

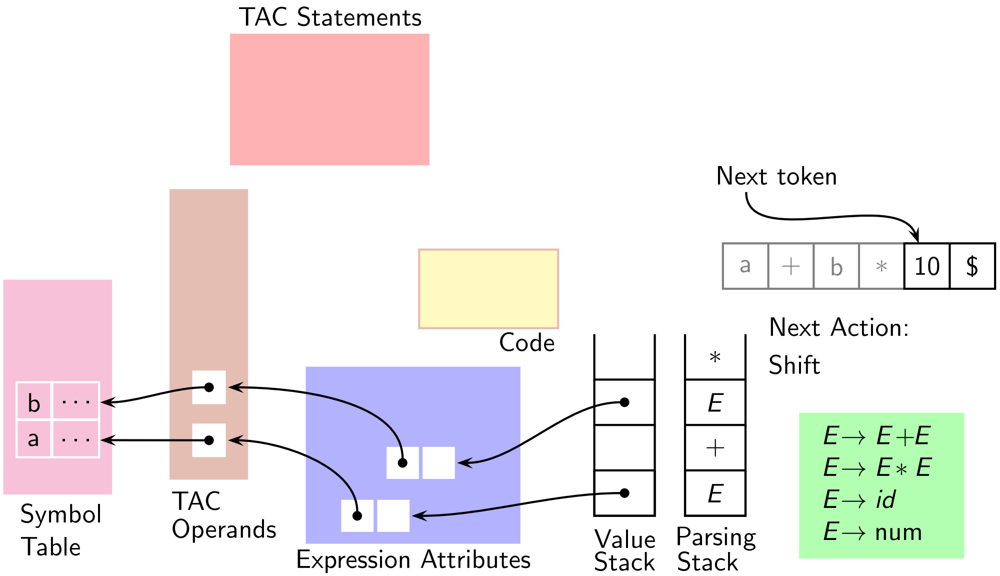
Lex & Yacc

Section:

Constructing TAC Statements During Parsing

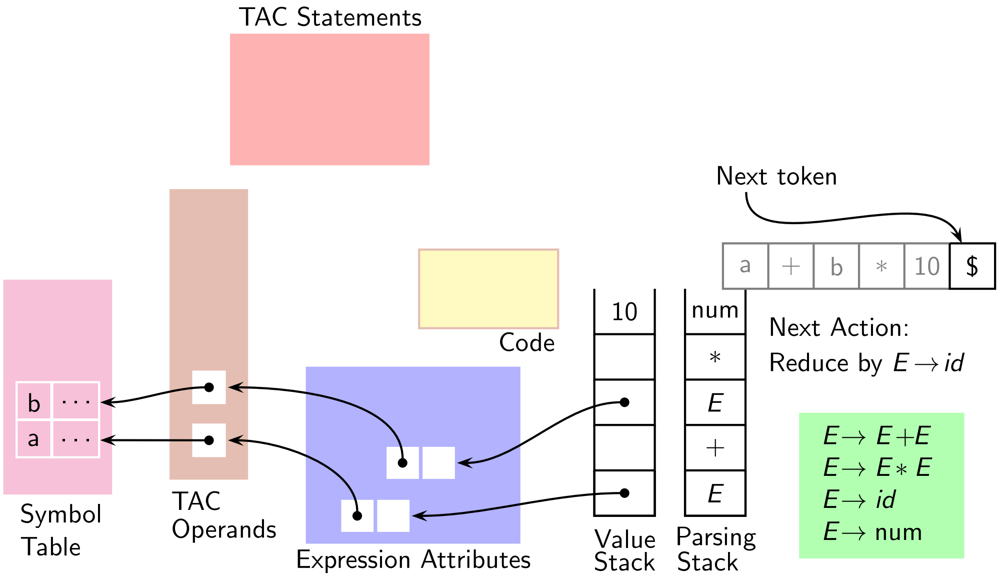


Constructing TAC Statements During Parsing

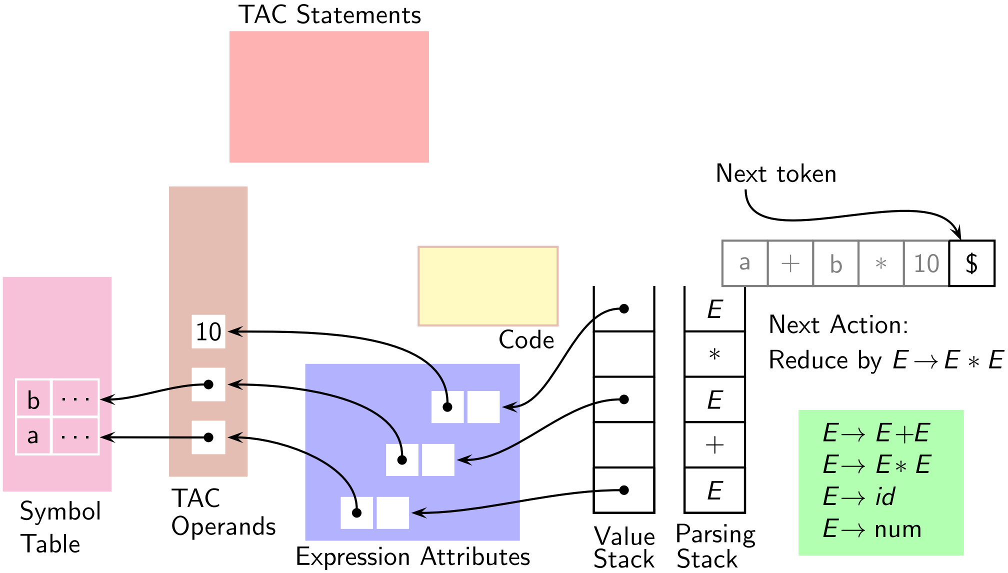


$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

Constructing TAC Statements During Parsing



Constructing TAC Statements During Parsing

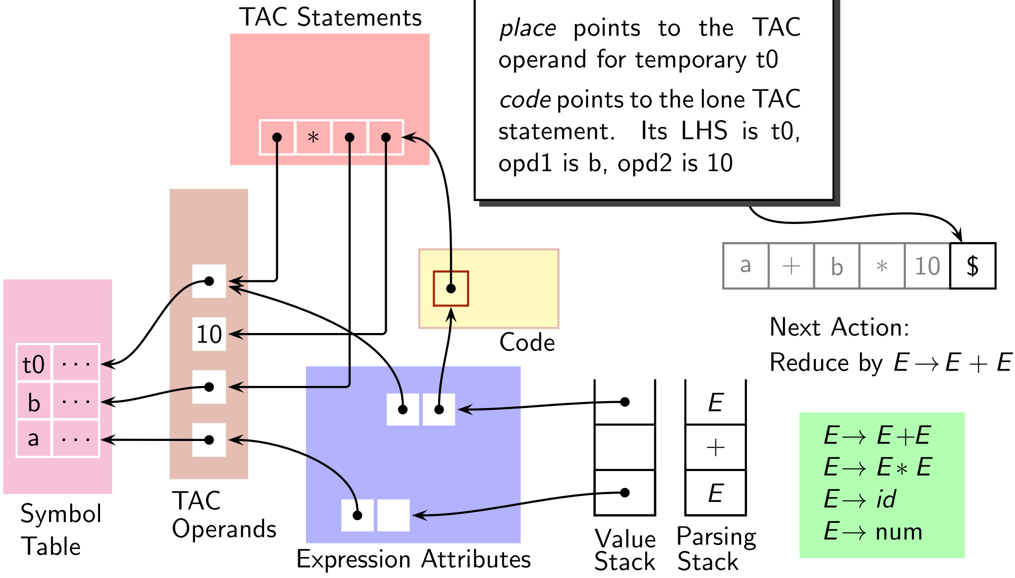


Next Action:

Reduce by $E \to E * E$

$E \to E + E$
$E \to E * E$
$E \to id$
$E \to \text{num}$

Constructing TAC Statements During Parsing



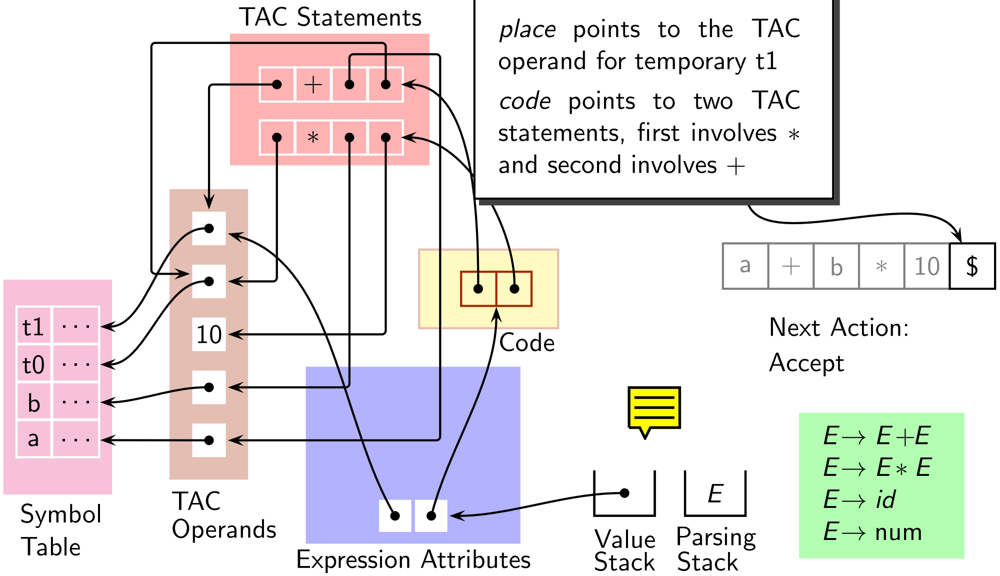
place points to the TAC operand for temporary t0

code points to the lone TAC statement. Its LHS is t0, opd1 is b, opd2 is 10

Next Action:
Reduce by $E \rightarrow E + E$

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

Constructing TAC Statements During Parsing



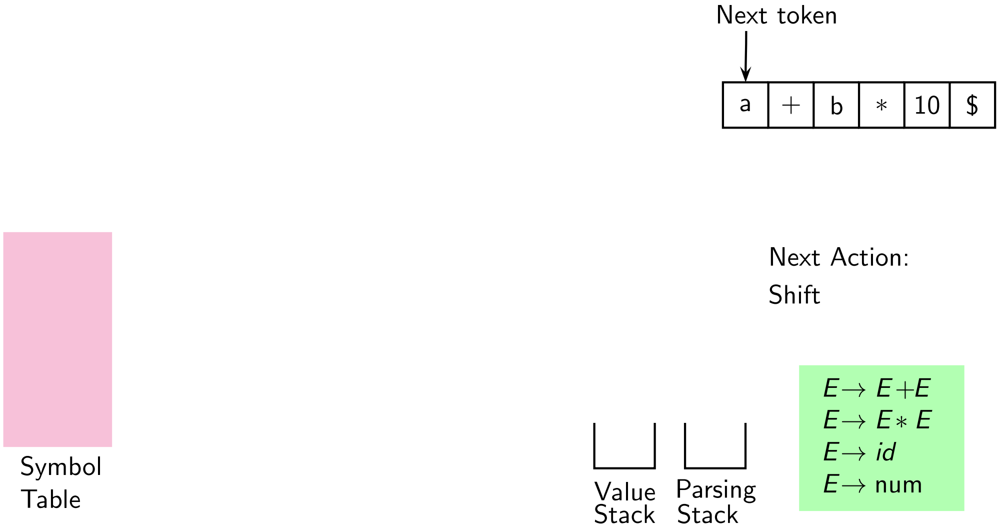
place points to the TAC operand for temporary t1

code points to two TAC statements, first involves $*$ and second involves $+$

Next Action:
Accept

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

Constructing ASTs During Parsing



$E \to E + E$
$E \to E * E$
$E \to id$
$E \to \text{num}$

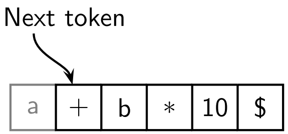
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

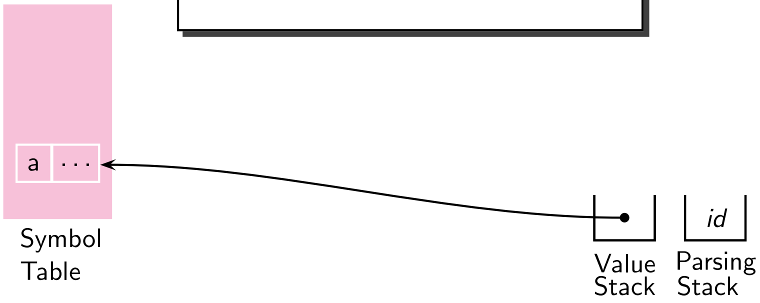
Section:

Constructing ASTs During Parsing



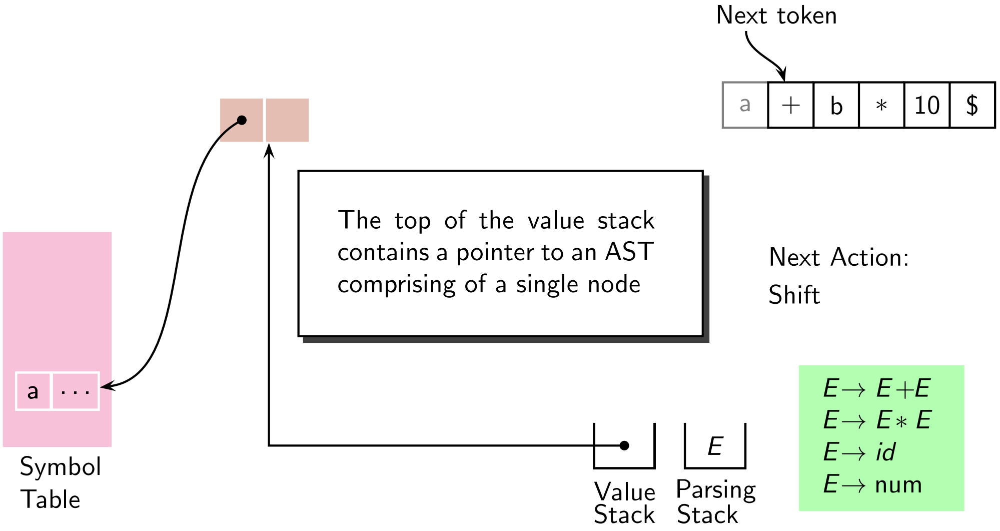
The top of the value stack contains a pointer to the symbol table entry of the name of the variable

Next Action:
Reduce by $E \rightarrow id$



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

Constructing ASTs During Parsing



The top of the value stack contains a pointer to an AST comprising of a single node

Next Action: Shift

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

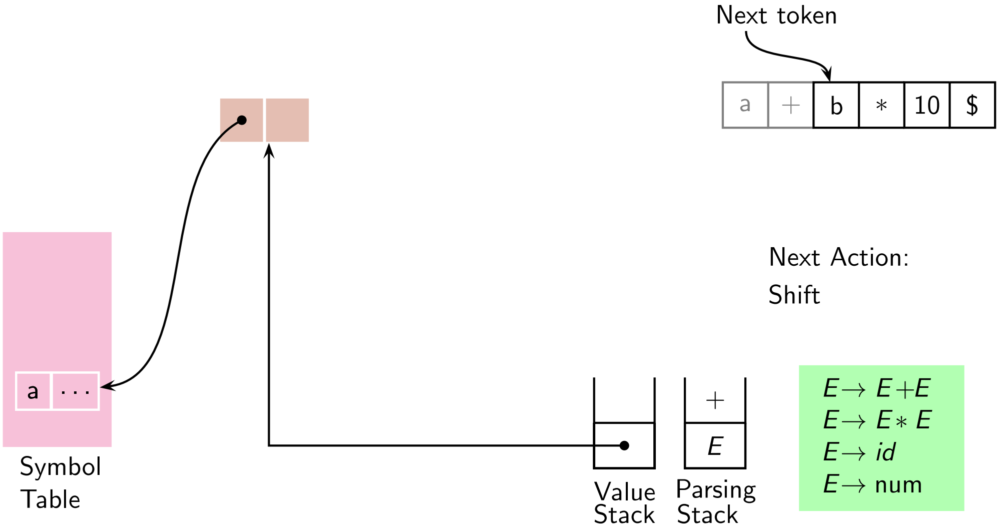
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

Constructing ASTs During Parsing



Next Action:
Shift

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

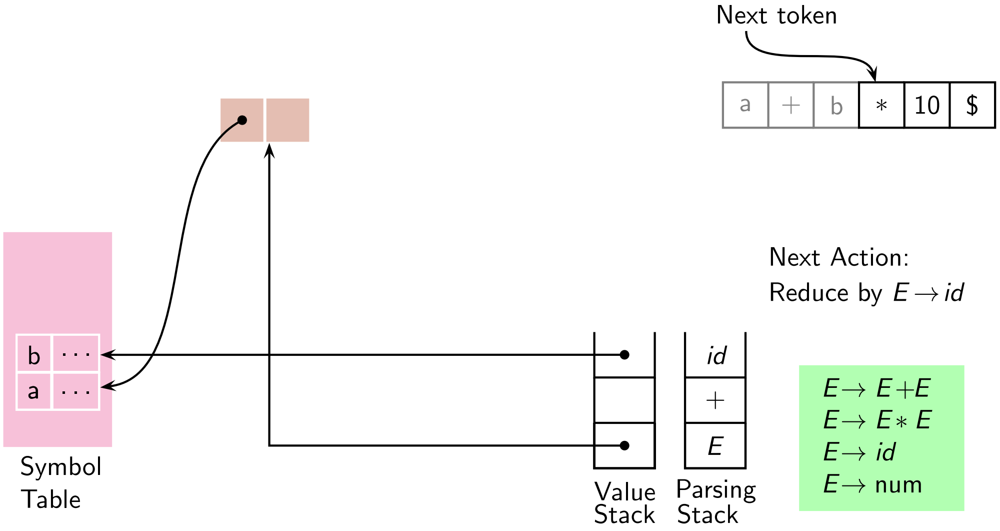
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

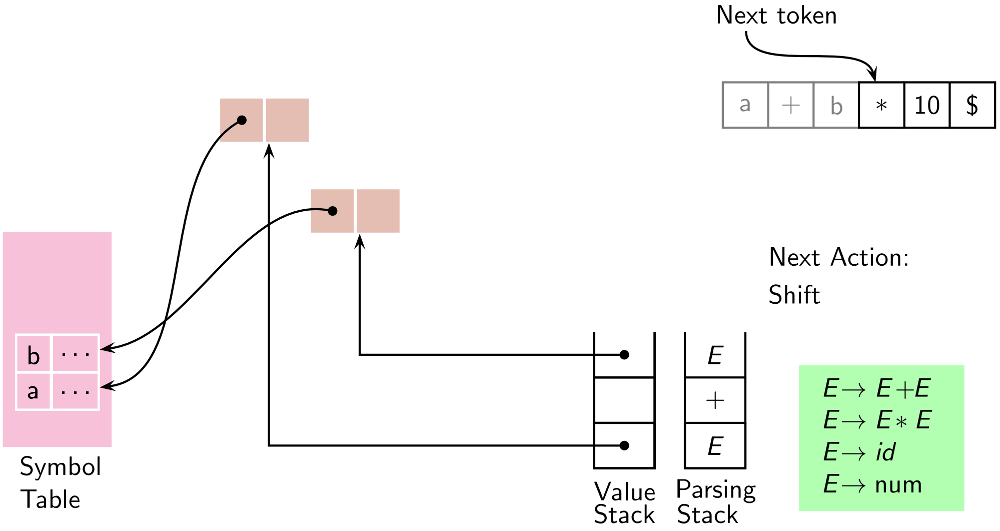
Constructing ASTs During Parsing



Next Action:
Reduce by $E \rightarrow id$

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

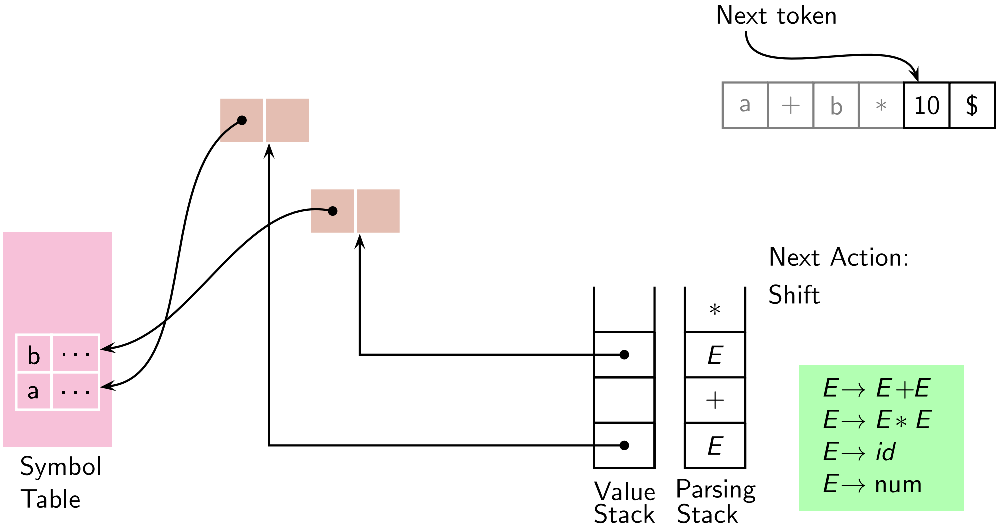
Constructing ASTs During Parsing



Next Action:
Shift

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

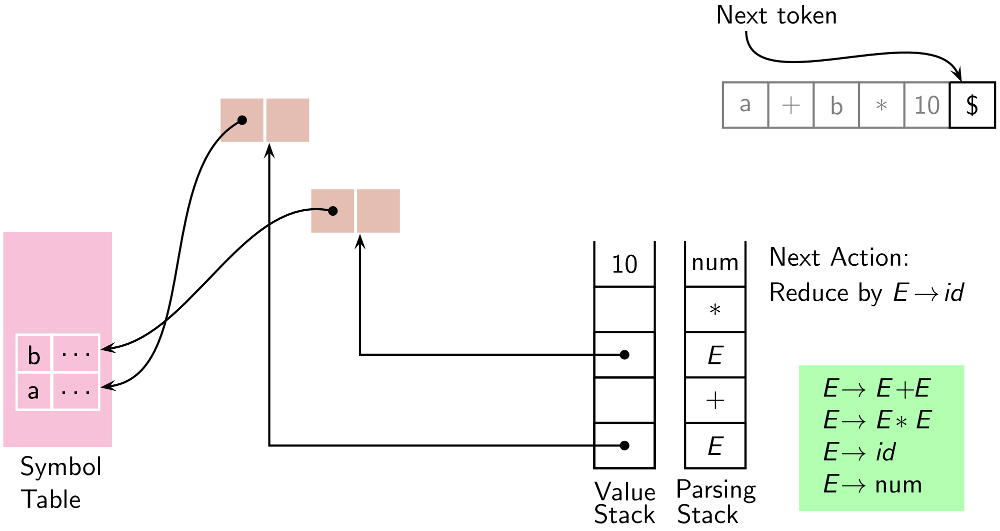
Constructing ASTs During Parsing



Next Action:
Shift

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

Constructing ASTs During Parsing



Next Action:
Reduce by $E \rightarrow id$

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

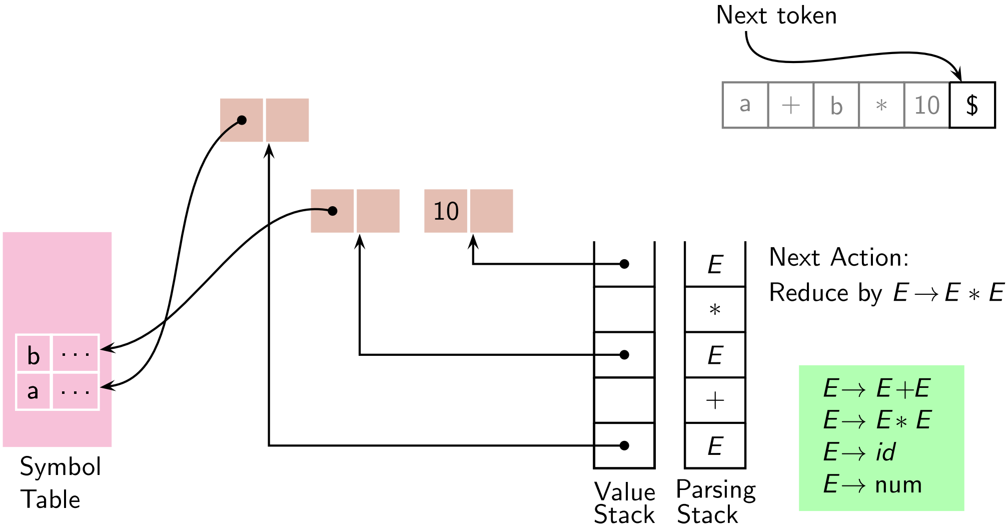
IIT Bombay
cs302: Implementation of Programming Languages

Topic:

Lex & Yacc

Section:

Constructing ASTs During Parsing

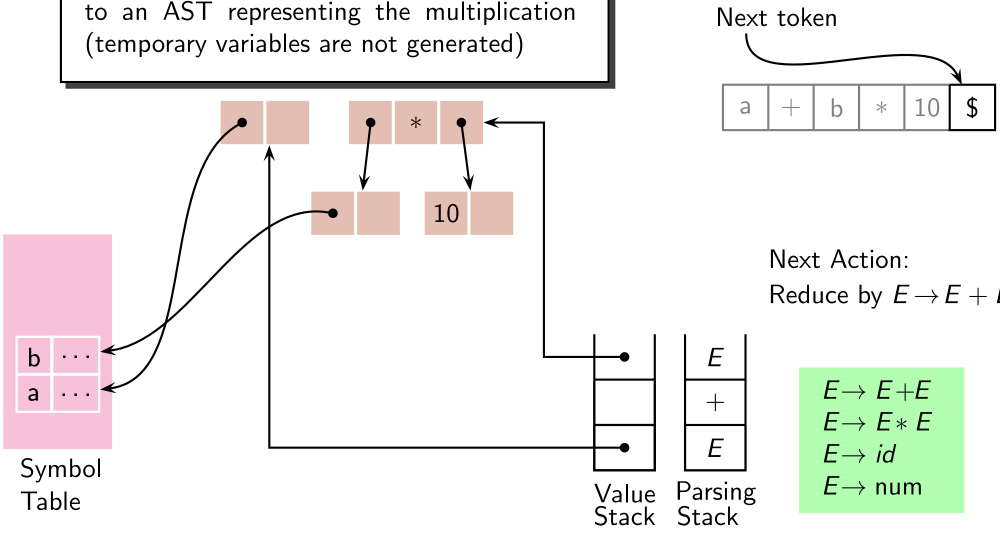


Next Action:
Reduce by $E \rightarrow E * E$

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

Constructing ASTs During Parsing

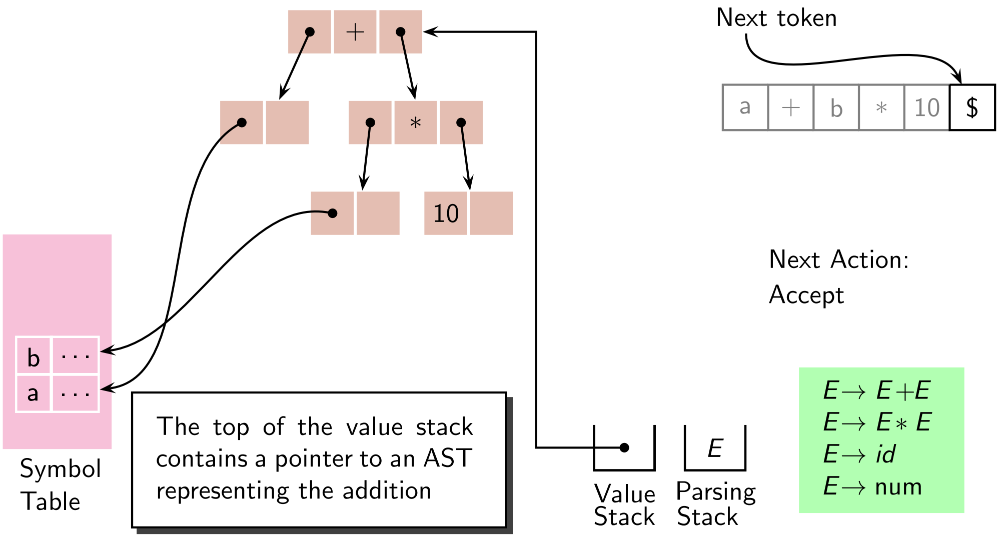
The top of the value stack contains a pointer to an AST representing the multiplication (temporary variables are not generated)



Next Action:
Reduce by $E \rightarrow E + E$

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

Constructing ASTs During Parsing



Next Action:
Accept

The top of the value stack contains a pointer to an AST representing the addition

$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num