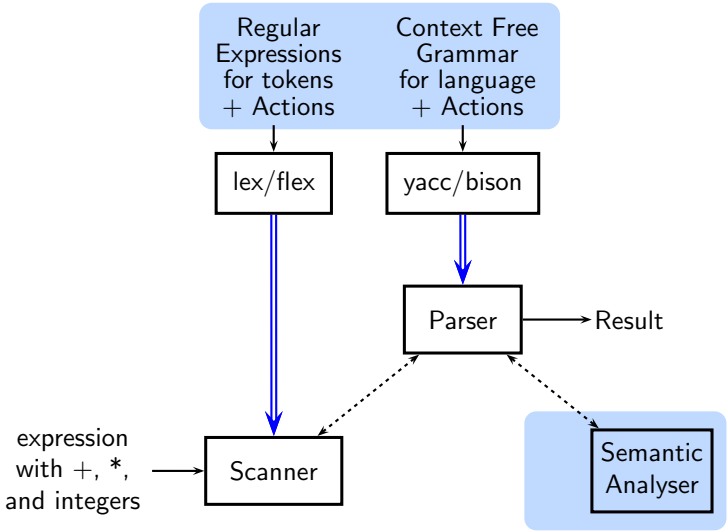
Introduction to Parsing Using Lex/Flex and Yacc/Bison

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

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

Jan 2025

simCalc: A Simple Calculator



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

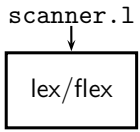
- Grammar

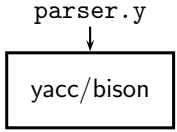
```
Expr  :  Expr '+' Expr
      |  Expr '*' Expr
      |  NUM
```

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

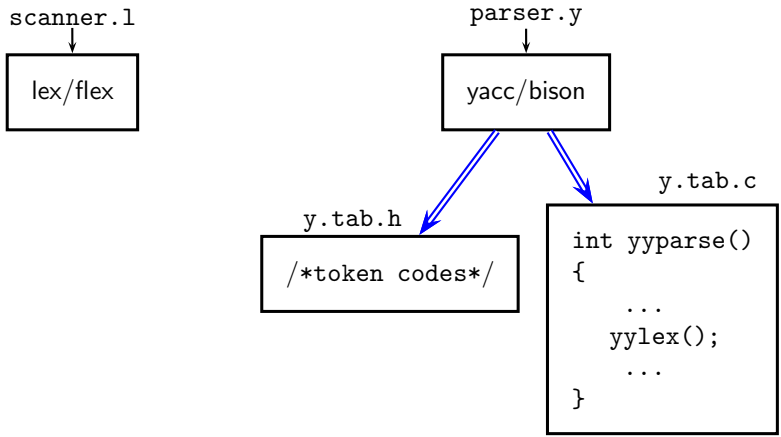
Token Name	Token Lexemes	Token Value	Token Code
Number	"10" "345" "03"	10 345 3	NUM
+ operator	"+"		'+'
* operator	"*"		'*'

The Interaction Between Scanner and Parser

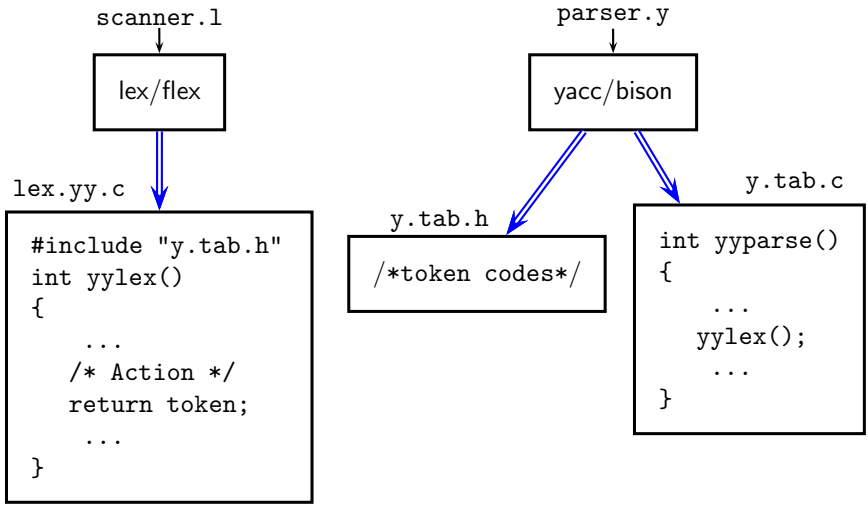




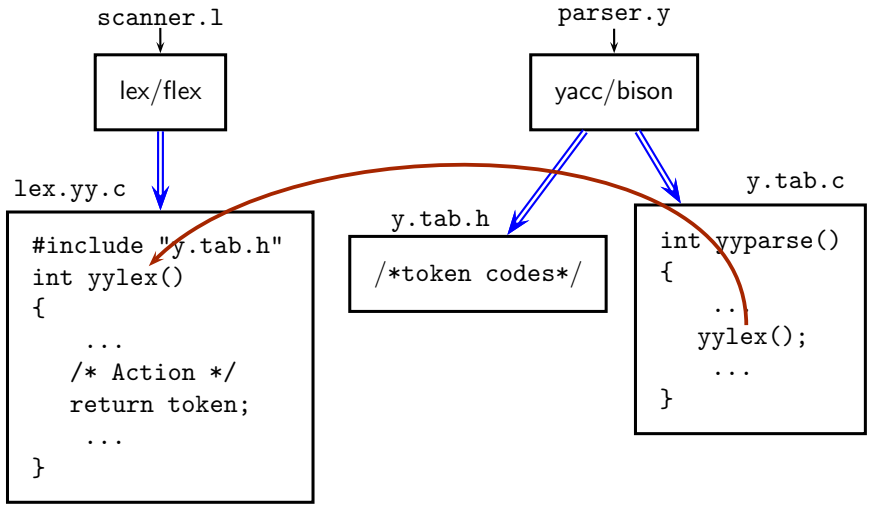
The Interaction Between Scanner and Parser



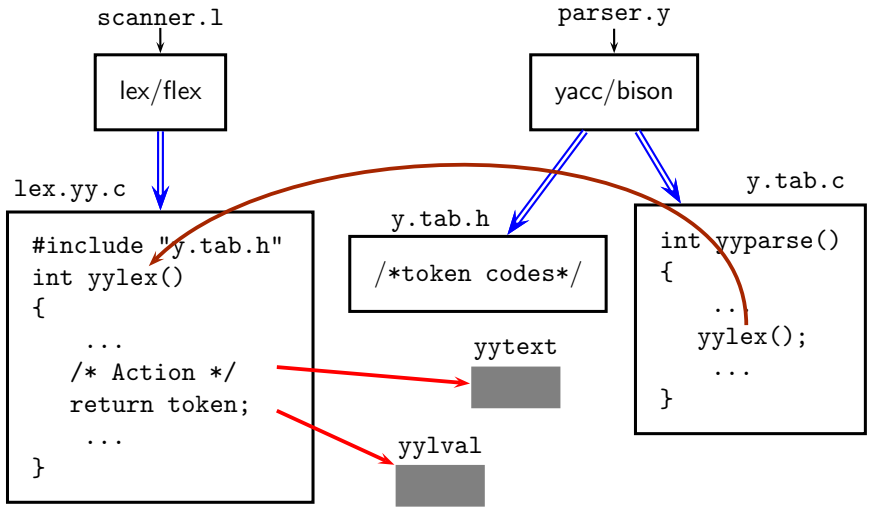
The Interaction Between Scanner and Parser



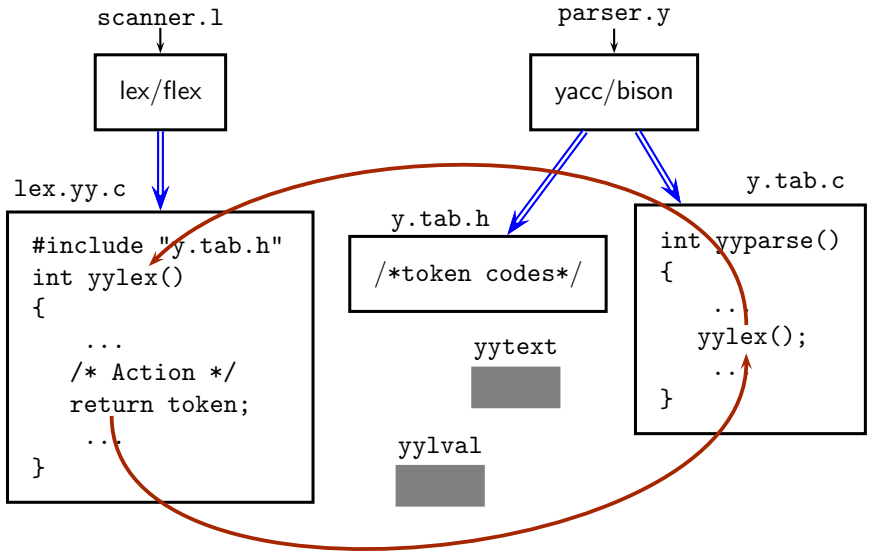
The Interaction Between Scanner and Parser



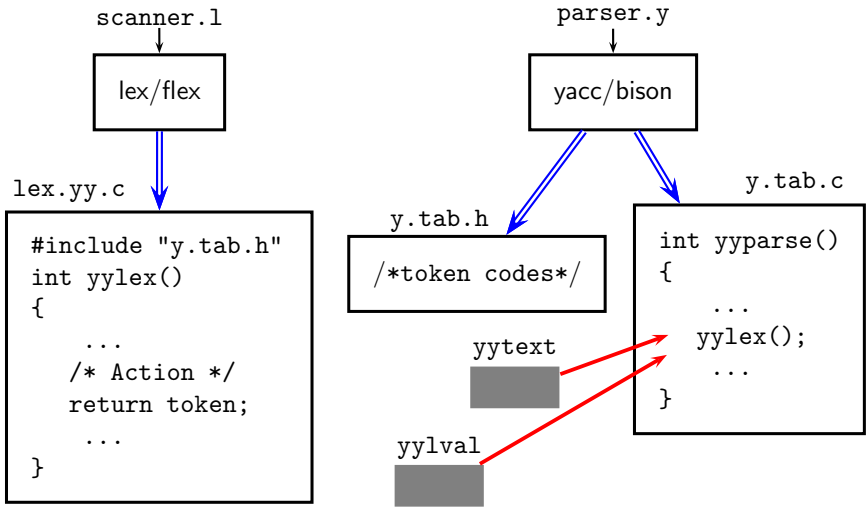
The Interaction Between Scanner and Parser



The Interaction Between Scanner and Parser



The Interaction Between Scanner and Parser



An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing

10	+	20	*	3	eof
NUM					

Parsing Stack

Action: Shift NUM

An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing

10 + 20 * 3 eof

+

top → | NUM |

Parsing Stack

Action: Reduce by "Expr: NUM"

An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing

10 + 20 * 3 eof

+

top → | Expr |

Parsing Stack

Action: Shift +

An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing

10	+	20	*	3	eof
		NUM			

top → | + |
| Expr |

Parsing Stack

Action: Shift NUM

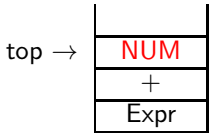
An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing

10	+	20	*	3	eof
			*		



Action: Reduce by "Expr: NUM"

An Overview of Shift Reduce Parsing

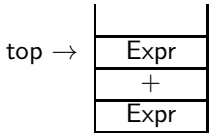
- Grammar

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

- The process of parsing

10 + 20 * 3 eof

*



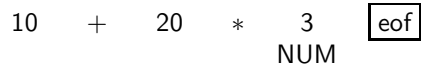
Action: Shift $*$ or Reduce by "Expr: Expr $+$ Expr"?

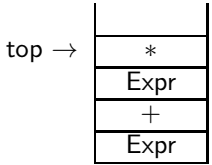
An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing





Action: Shift NUM

An Overview of Shift Reduce Parsing

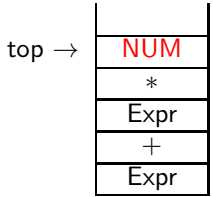
- Grammar

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

- The process of parsing

10 + 20 * 3 eof

EOF



Action: Reduce by "Expr: NUM"

An Overview of Shift Reduce Parsing

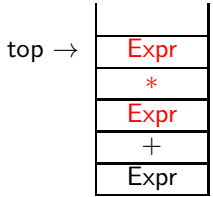
- Grammar

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

- The process of parsing

10 + 20 * 3 eof

EOF



Action: Reduce by "Expr: Expr * Expr"

An Overview of Shift Reduce Parsing

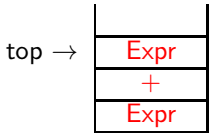
- Grammar

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

- The process of parsing

10 + 20 * 3 eof

EOF



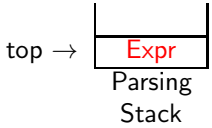
Action: Reduce by "Expr: Expr + Expr"

An Overview of Shift Reduce Parsing

- Grammar

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

- The process of parsing



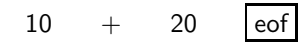
Action: Accept

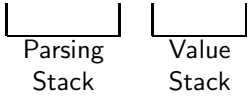
An Overview of Attribute Evaluation

- 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

10 + 20 eof

+

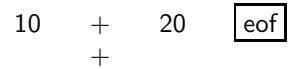
top → NUM (Parsing Stack) | 10 (Value Stack) $1

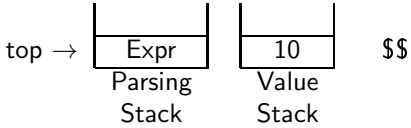
An Overview of Attribute Evaluation

- Grammar and the associated actions

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

- The process of parsing and attribute evaluation



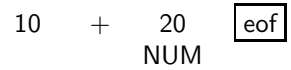


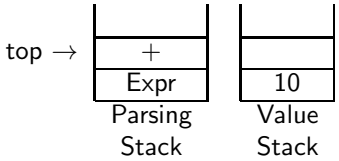
An Overview of Attribute Evaluation

- Grammar and the associated actions

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

- The process of parsing and attribute evaluation





An Overview of Attribute Evaluation

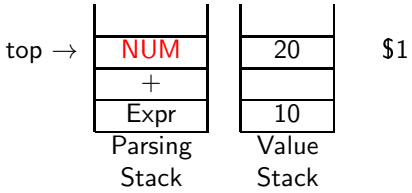
- Grammar and the associated actions

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

- The process of parsing and attribute evaluation

10 + 20 eof

EOF



An Overview of Attribute Evaluation

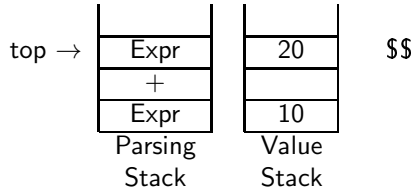
- Grammar and the associated actions

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

- The process of parsing and attribute evaluation

10 + 20 eof

EOF

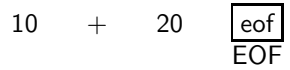


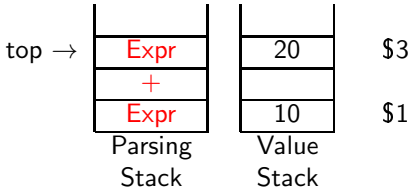
An Overview of Attribute Evaluation

- Grammar and the associated actions

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

- The process of parsing and attribute evaluation



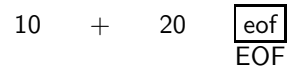


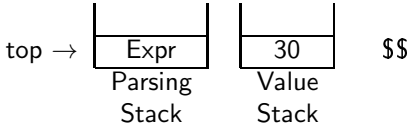
An Overview of Attribute Evaluation

- Grammar and the associated actions

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

- The process of parsing and attribute evaluation





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

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

Constructing TAC Statements During Parsing

TAC Statements

Symbol Table

TAC Operands

Code

Expression Attributes

Next token

a	+	b	*	10	$

Next Action:
Shift

Value Stack

Parsing Stack

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

Constructing TAC Statements During Parsing

TAC Statements

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

Code

Next token

a	+	b	*	10	$

Next Action:
Reduce by $E \rightarrow id$

a	···

Symbol Table

TAC Operands

Expression Attributes

Value Stack

Parsing Stack

id

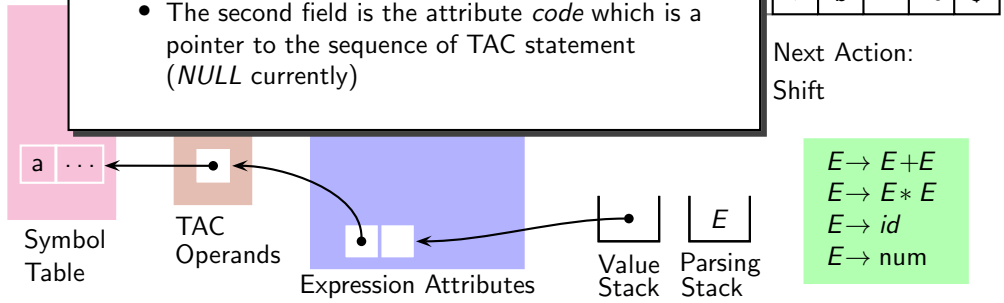
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

Constructing TAC Statements During Parsing

TAC Statements

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

TAC Statements

Next token

a	+	b	*	10	$

Next Action:
Shift

Code

a	...

Symbol Table

TAC Operands

Expression Attributes

Value Stack

Parsing Stack

+
E

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

Constructing TAC Statements During Parsing

TAC Statements

Next token

a	+	b	*	10	$

Next Action:
Reduce by $E \rightarrow id$

Code

b	···
a	···

Symbol Table

TAC Operands

Expression Attributes

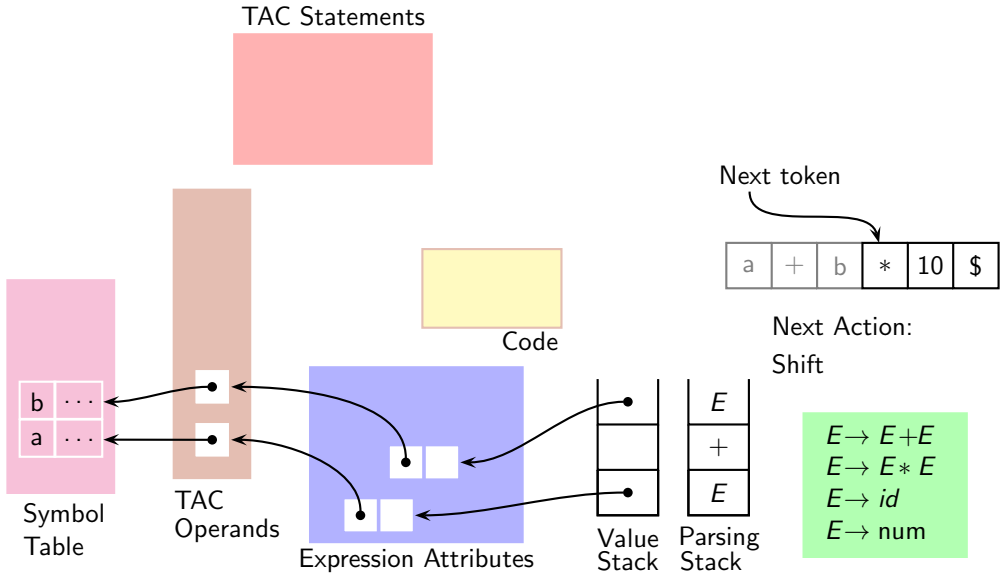
Value Stack

id
+
E

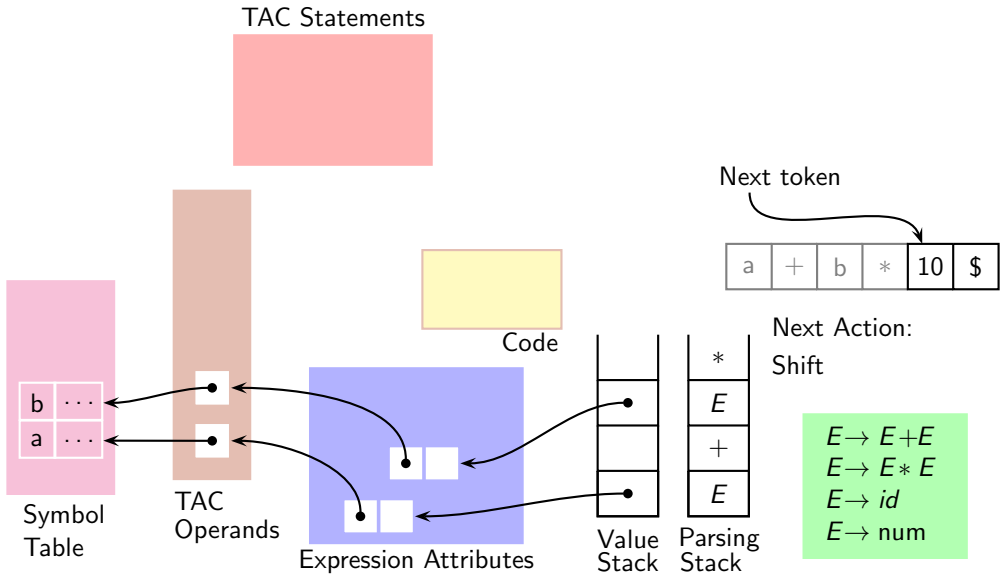
Parsing Stack

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

Constructing TAC Statements During Parsing



Constructing TAC Statements During Parsing



Constructing TAC Statements During Parsing

TAC Statements

Next token

a	+	b	*	10	$

Code

b	···
a	···

Symbol Table

TAC Operands

Expression Attributes

10

Value Stack

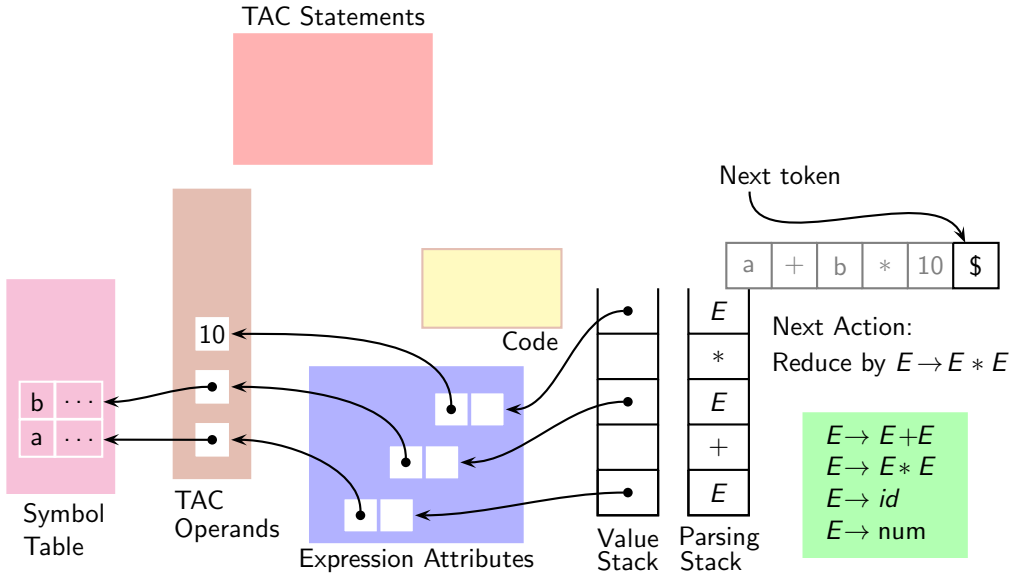
num
*
E
+
E

Parsing Stack

Next Action:
Reduce by $E \rightarrow id$

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

Constructing TAC Statements During Parsing



Constructing TAC Statements During Parsing

TAC Statements

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

a + b * 10 $

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

t0 ···
b ···
a ···

Symbol Table

10

TAC Operands

Code

Expression Attributes

Value Stack

Parsing Stack

E
+
E

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

Constructing TAC Statements During Parsing

TAC Statements

place points to the TAC operand for temporary t1

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

a | + | b | * | 10 | $

Next Action:
Accept

t1 ···
t0 ···
b ···
a ···

Symbol Table

10

TAC Operands

Code

Expression Attributes

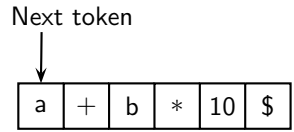
Value Stack

Parsing Stack

E

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

Constructing ASTs During Parsing



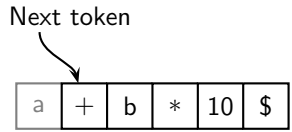
Next Action:
Shift

Symbol Table

Value Stack Parsing Stack

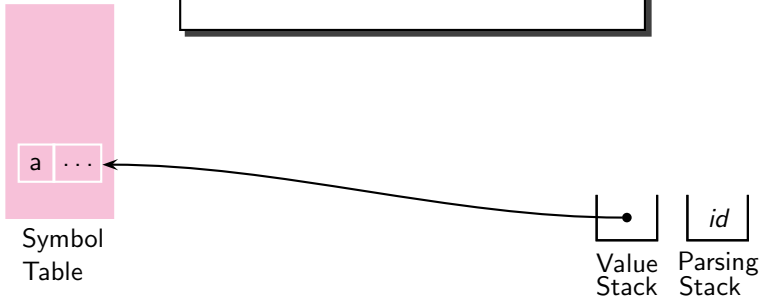
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

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 $E \rightarrow id$



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

Constructing ASTs During Parsing

Next token

a	+	b	*	10	$

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

Next Action:
Shift

a	...

Symbol Table

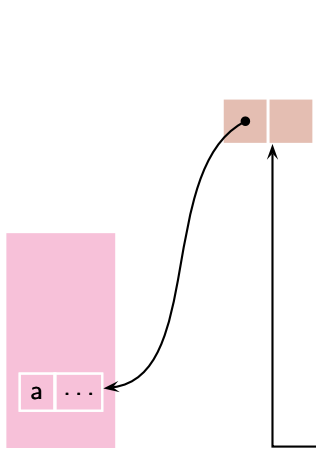
Value Stack

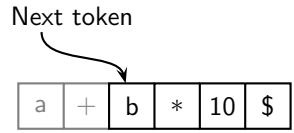
E

Parsing Stack

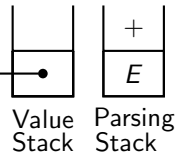
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

Constructing ASTs During Parsing



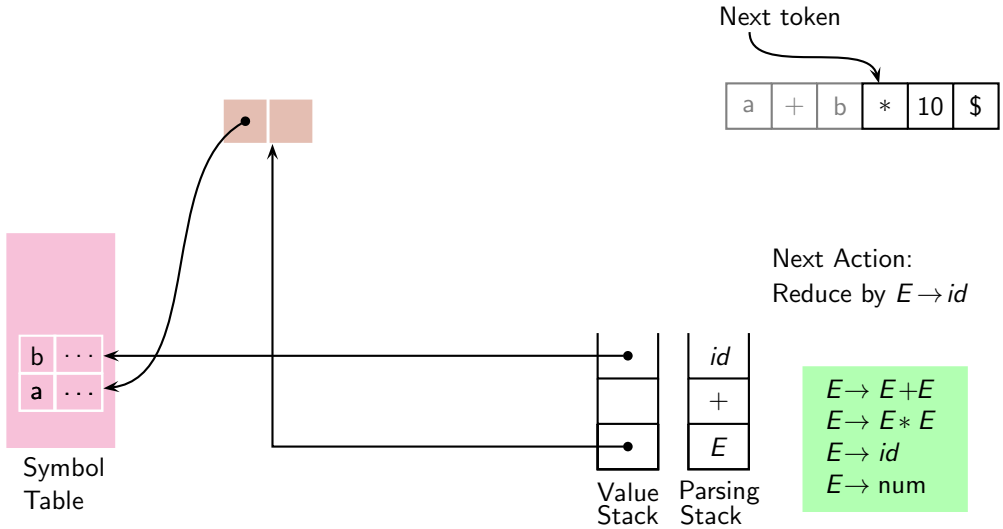


Next Action:
Shift

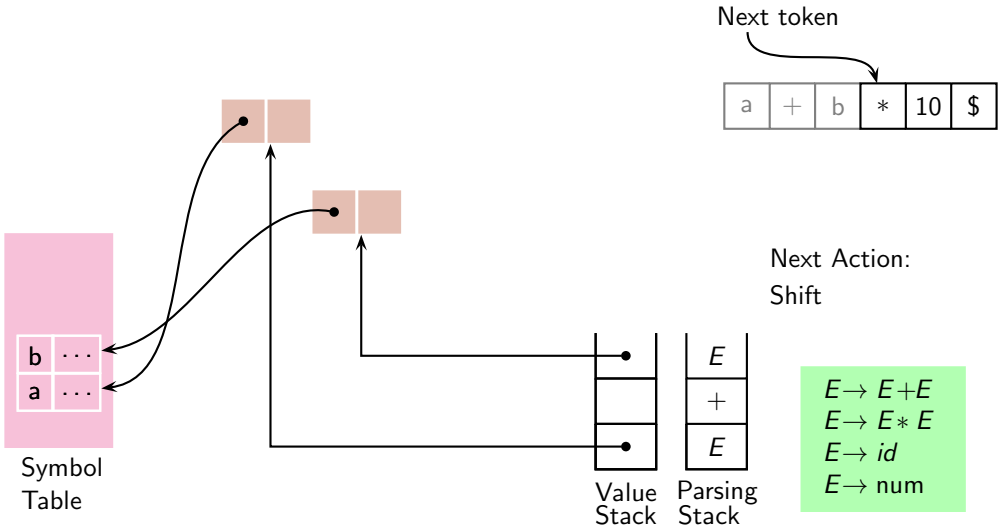


$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

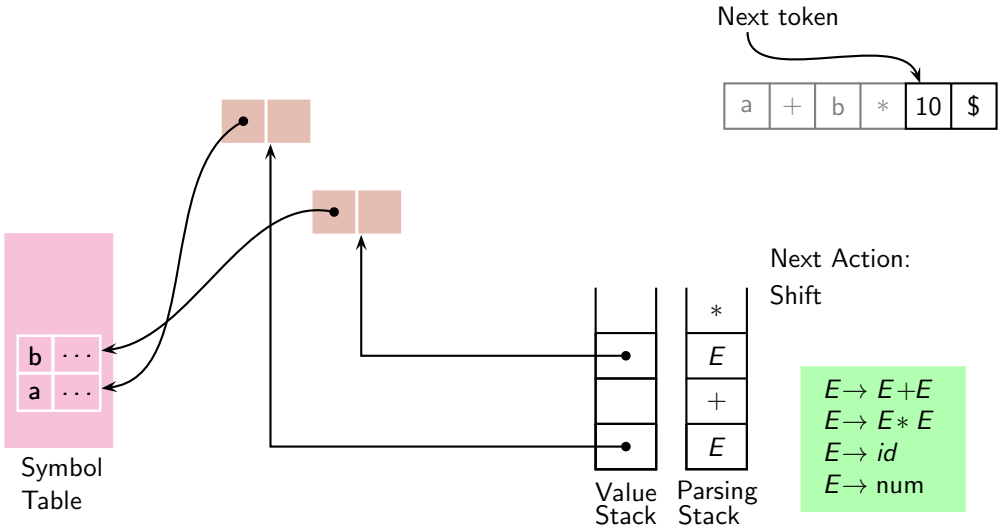
Constructing ASTs During Parsing



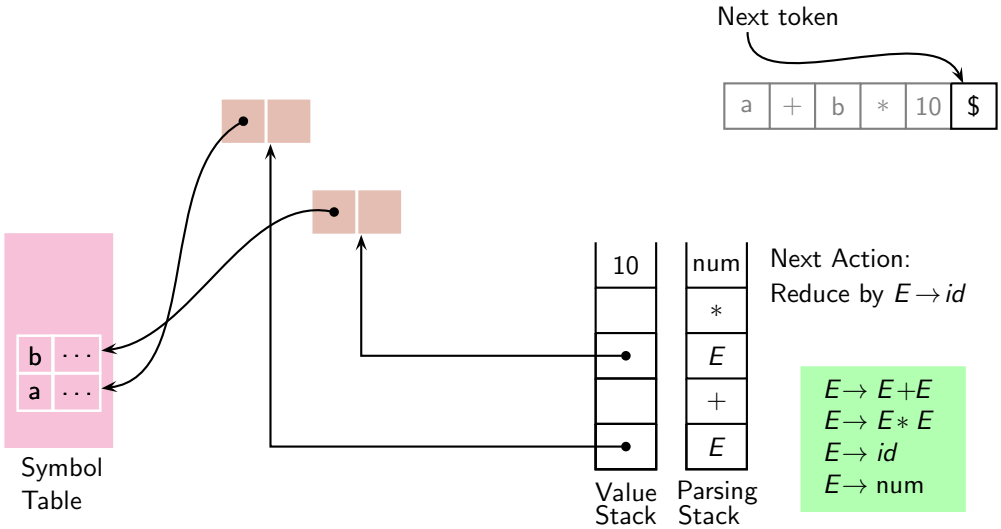
Constructing ASTs During Parsing



Constructing ASTs During Parsing



Constructing ASTs During Parsing



Constructing ASTs During Parsing

Next token

a	+	b	*	10	$

b	...
a	...

Symbol Table

10

E
*
E
+
E

Value Stack

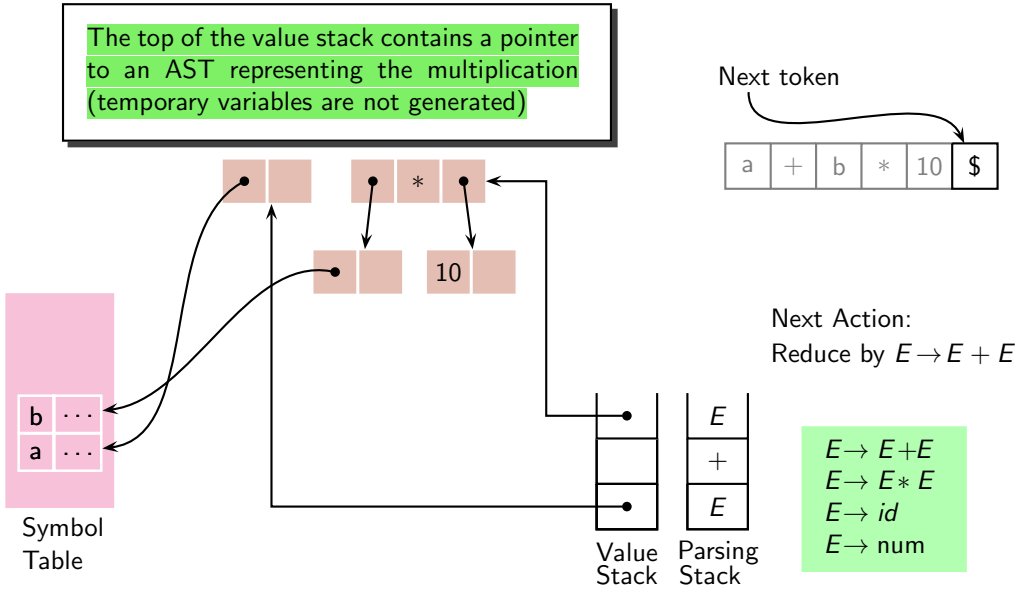
Parsing Stack

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

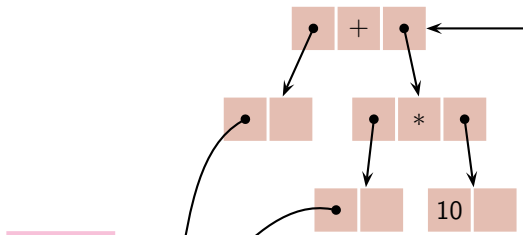
$E \rightarrow E + E$
$E \rightarrow E * E$
$E \rightarrow id$
$E \rightarrow$ num

Constructing ASTs During Parsing

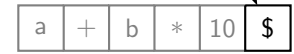
The top of the value stack contains a pointer to an AST representing the multiplication (temporary variables are not generated)



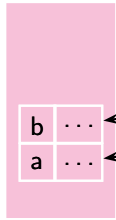
Constructing ASTs During Parsing



Next token



Next Action:
Accept



Symbol Table

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

Value Stack Parsing Stack

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