BITS Pilani
Pilani Campus

Semantic Analysis

Dr. Shashank Gupta
Assistant Professor
Department of Computer Science and Information Systems

Motivation

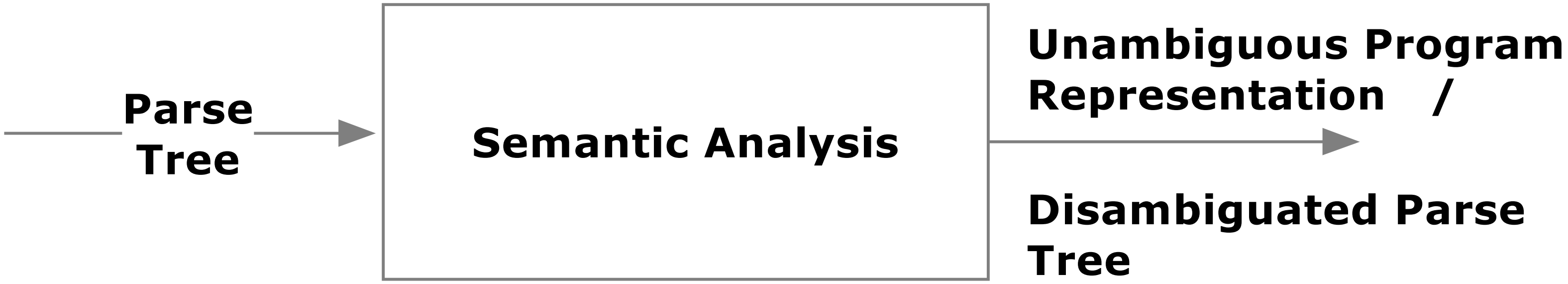
Parser cannot catch all the program errors.

- There is a level of correctness that is deeper than syntax analysis.

Some language features cannot be modeled using context free grammar formalism

- Whether an identifier has been declared before use

Semantic Analysis



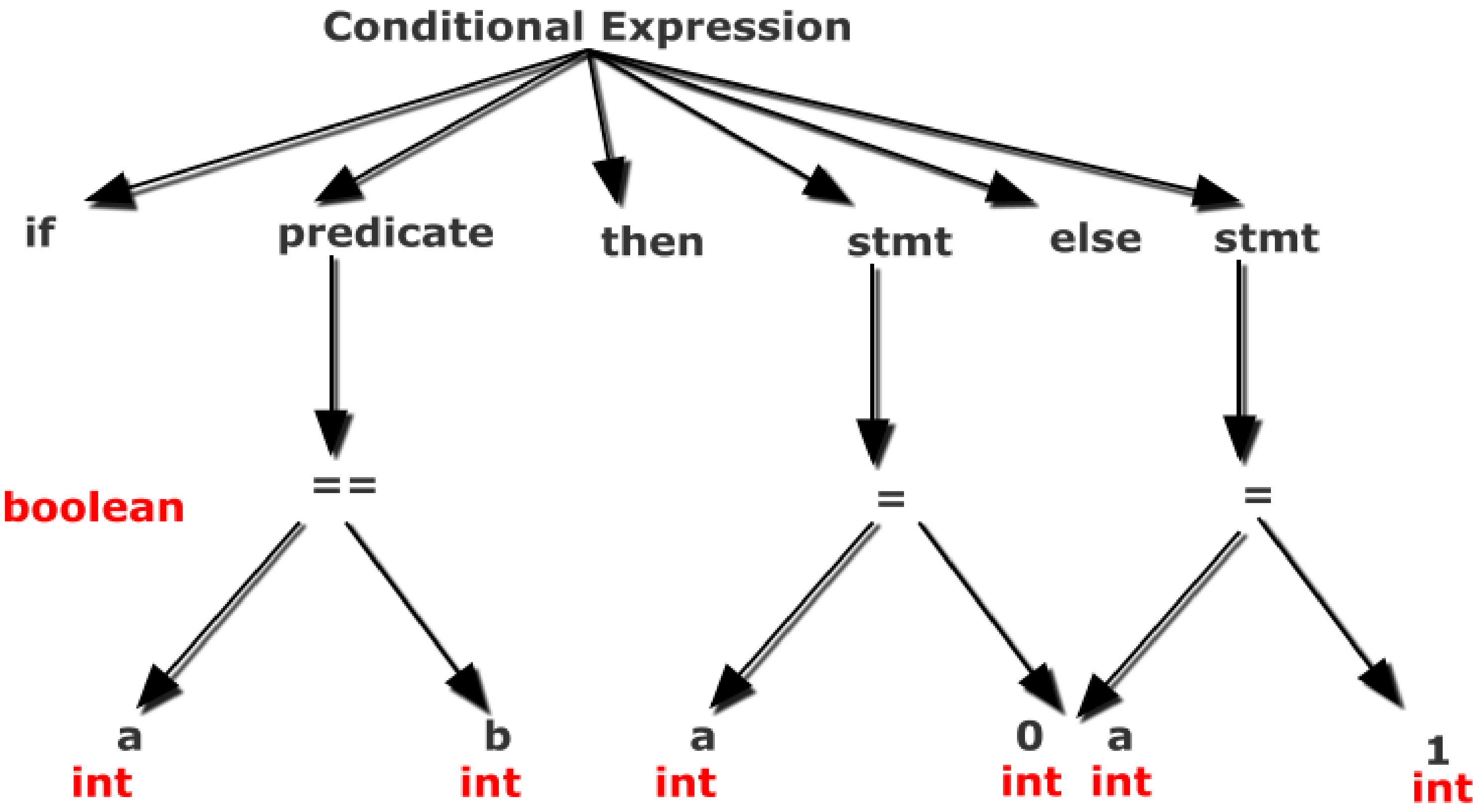
Parse Tree

Semantic Analysis

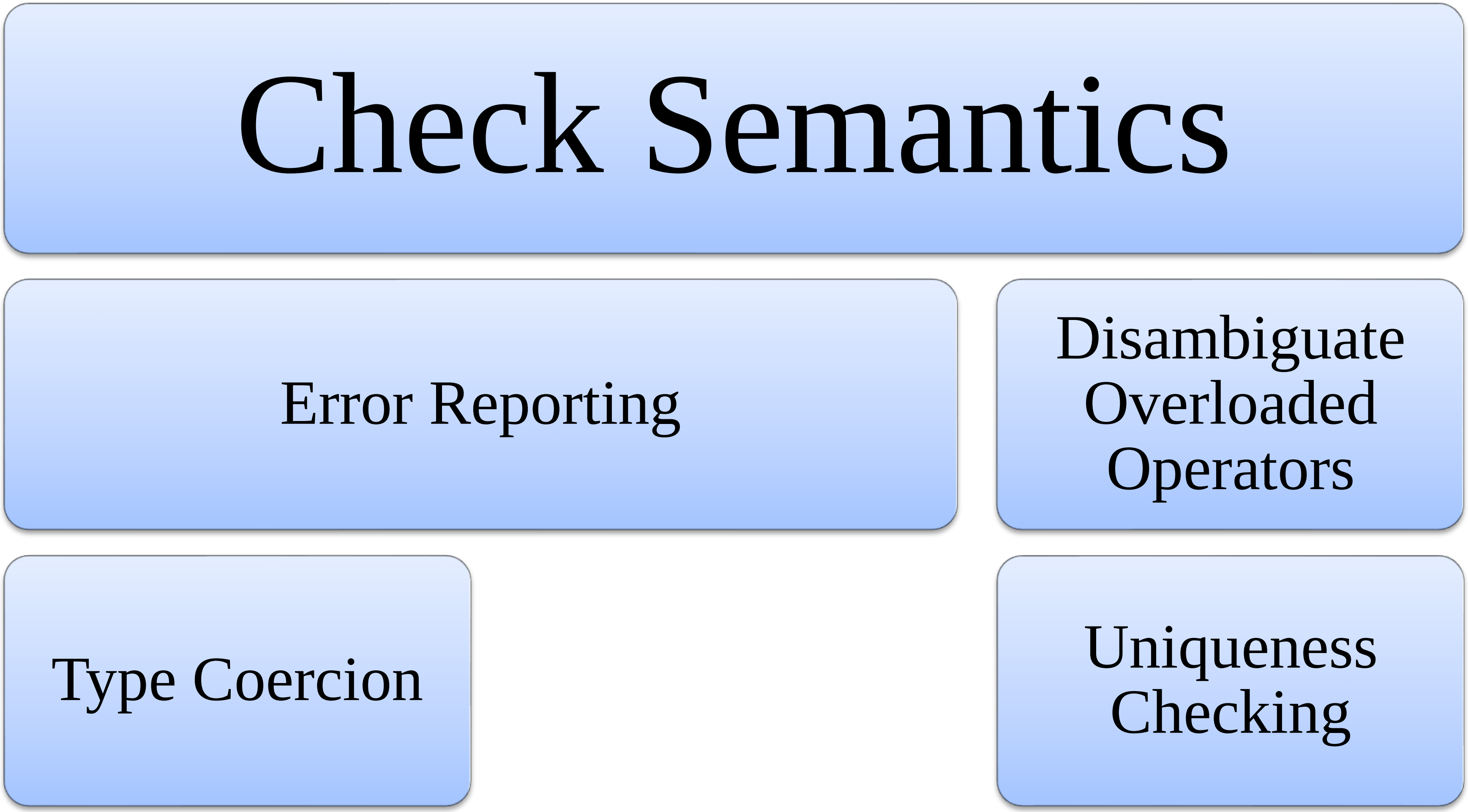
Unambiguous Program Representation /

Disambiguated Parse Tree

Output of Semantic Analysis: Disambiguated Parse Tree



More on Semantic Analysis



Check Semantics

Error Reporting

Disambiguate Overloaded Operators

Type Coercion

Uniqueness Checking

Semantic Errors

```
String x; int y;
y = x + 3;
```

The use of x could be a type error

```
int a, b;
a = b + c;
```

Here, c is not declared

An identifier may refer to different variables in different parts of the program

An identifier may be usable in one part of the program but not another

Requirements of Semantic Analysis

- Whether a variable has been declared?
- Are there variables which have not been declared?
- What is the type of the variable?
- Whether a variable is a scalar, an array, or a function?
- What declaration of the variable does each reference use?
- If an expression is type consistent?

Requirements of Semantic Analysis

- How many arguments does a function take?
- Are all invocations of a function consistent with the declaration?
- If an operator/function is overloaded, which function is being invoked?
- **The exact requirements depend upon the language.**

How to Address these Requirements?

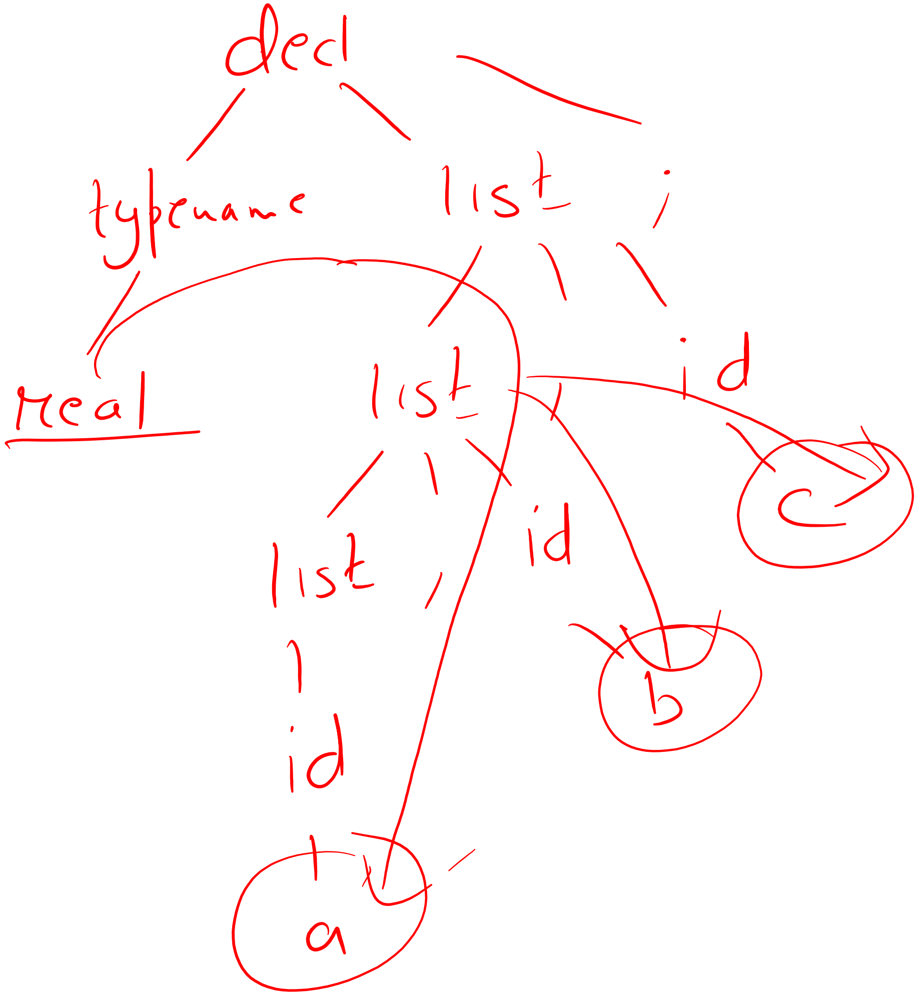
Answers to these questions depend upon values like type information, number of parameters etc.

- Compiler will have to do some computation to arrive at answers

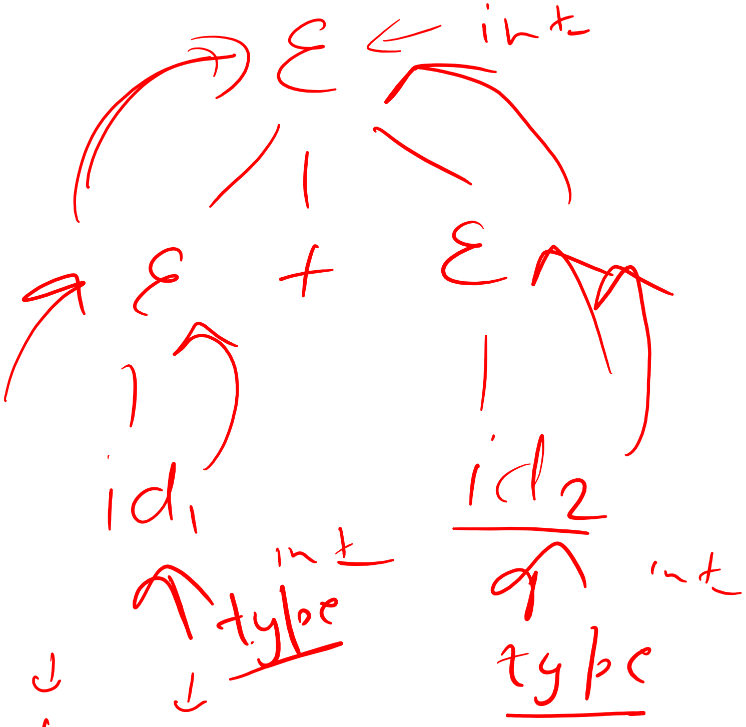
Do analysis along with parsing using attributes (also known as **Attribute Grammar Formalism**) real a, b, c;

decl → typename list ;

list → list , id | id



E → E + E | id



a = b + c ;

int str const

Attribute Grammar

Framework

3 cast | ε

$E \rightarrow id$, $E.type = id.type$

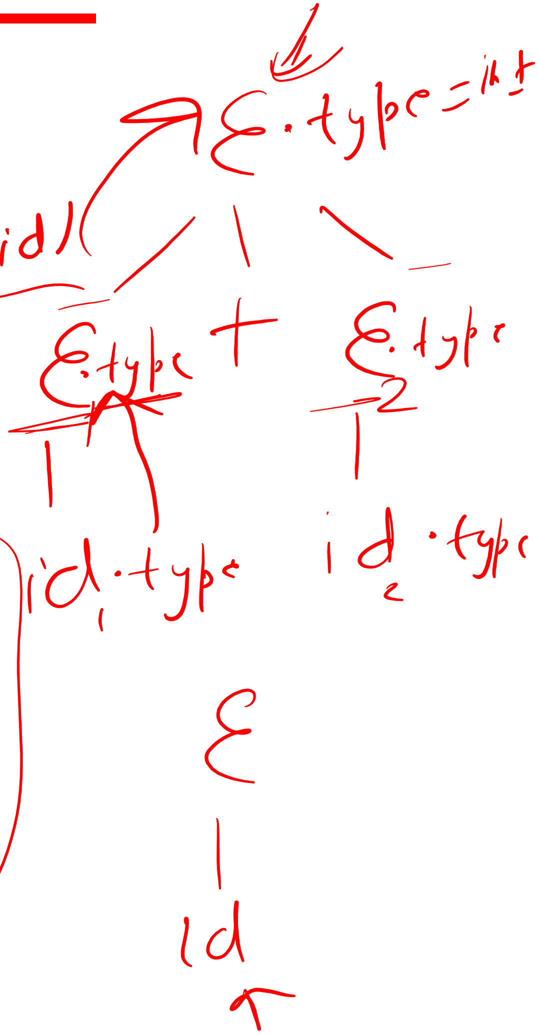
lookup.type(id)

$E \rightarrow E_1 + E_2$

$\{$ if $E_1.type == int$

$\&\&$ $E_2.type == int$

$E.type = int$ $\}$



Attribute Grammar Framework

Generalization of CFG where each grammar symbol has an associated set of attributes type

Values of attributes are computed by semantic rules.

Notations for associating Semantic Rules with Productions

Syntax Directed Definition

- High Level Specifications
- Hides implementation details
- Explicit order of evaluation is not specified

Translation Scheme

- Indicate order in which semantic rules are to be evaluated.
- Allow some implementation details to be shown.

Attribute Grammar Framework

Conceptually both

- Parse input token stream
- Build parse tree
- Traverse the parse tree to evaluate the semantic rules at the parse tree nodes

Evaluation may

- Save information in the symbol table
- Issue error messages
- Generate code
- Perform any other activity

Types of Attributes: Synthesized and Inherited

Value of a synthesized attribute is computed from the values of children nodes

- Attribute value for LHS of a rule comes from attributes of RHS

Value of an inherited attribute is computed from the sibling and parent nodes

- Attribute value for a symbol on RHS of a rule comes from attributes of LHS and RHS symbols

Attributes

Each production rule A → α has associated with it a set of semantic rules of the form

```
a = p (d1 , d2 , ..., dk )
```

- where p is a function, and

  Either a is a synthesized attribute of A

  OR a is an inherited attribute of one of the grammar symbols on the right

- Attribute a depends on attributes d1 , d2 , ..., dk

Synthesized Attributes

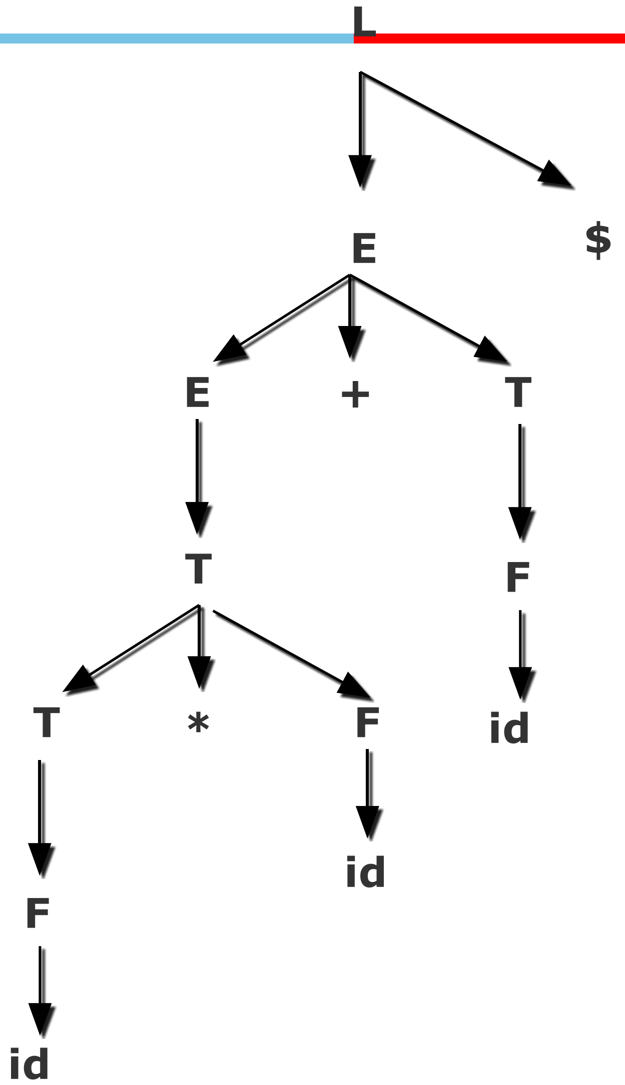
A syntax directed definition that uses only synthesized attributes is said to be an S-attributed definition

A parse tree for an S-attributed definition can be annotated by evaluating semantic rules for attributes

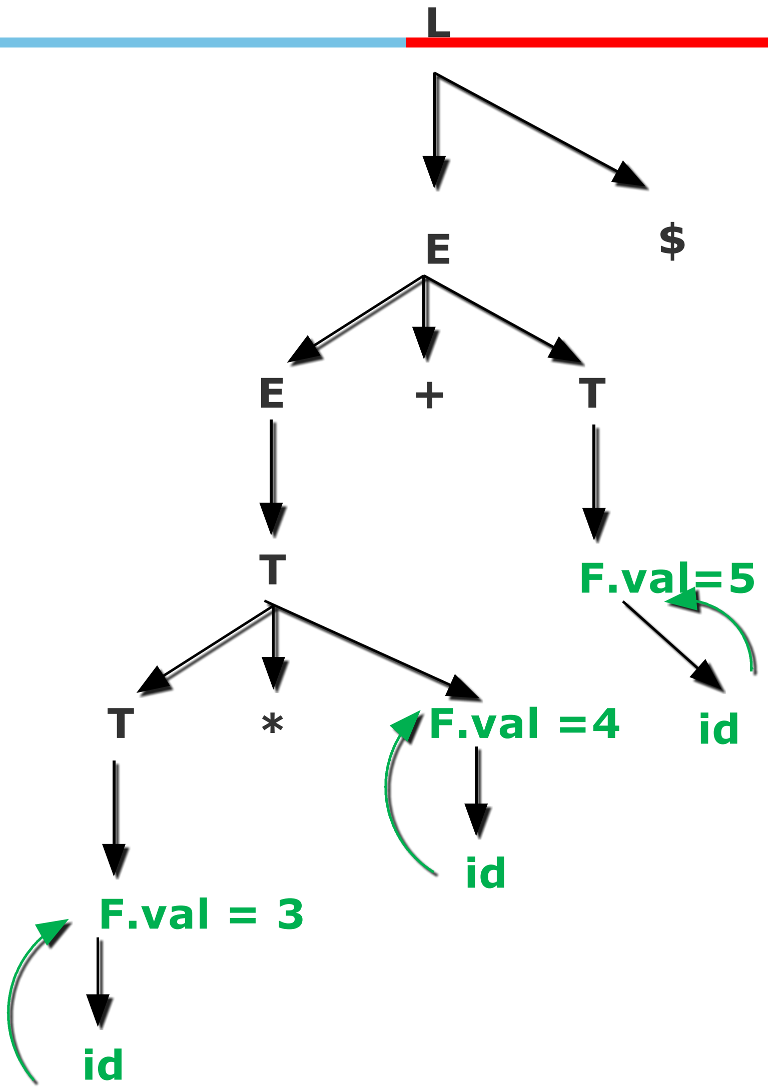
Syntax Directed Definitions for Given CFG

- L → E $
- E → E + T
- E → T
- T → T * F
- T → F
- F → (E)
- F → id

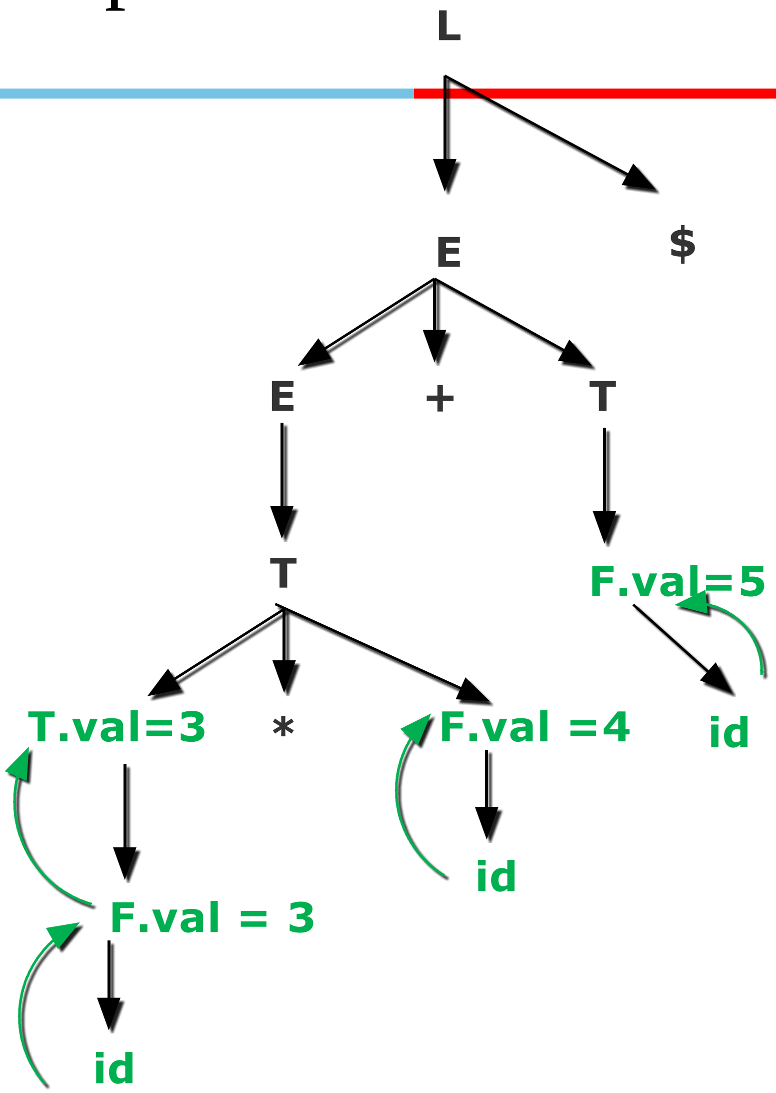
Parse Tree for 3*4+5



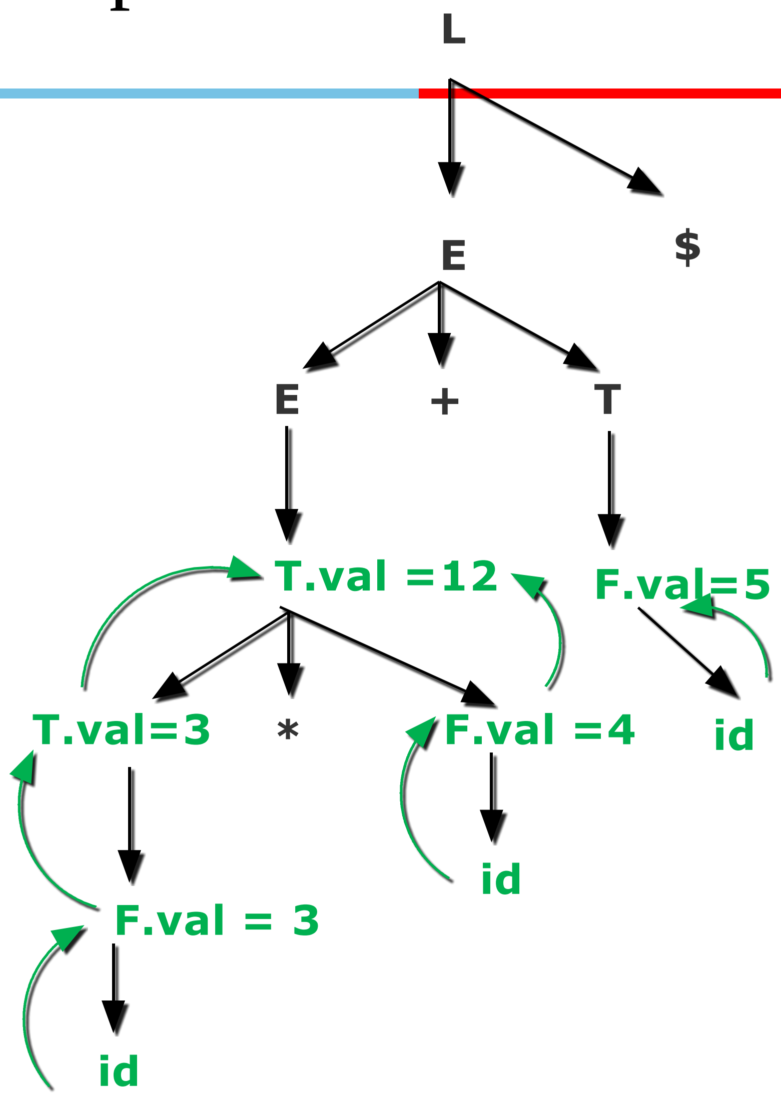
Dependence Graph



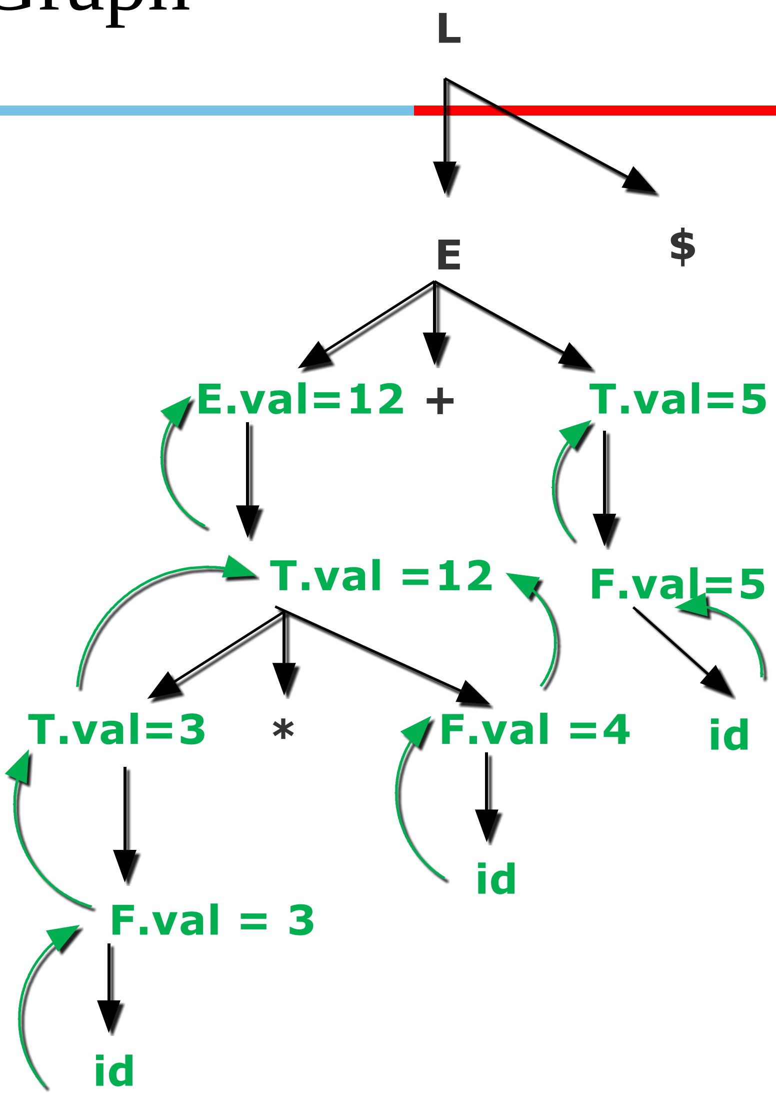
Dependence Graph



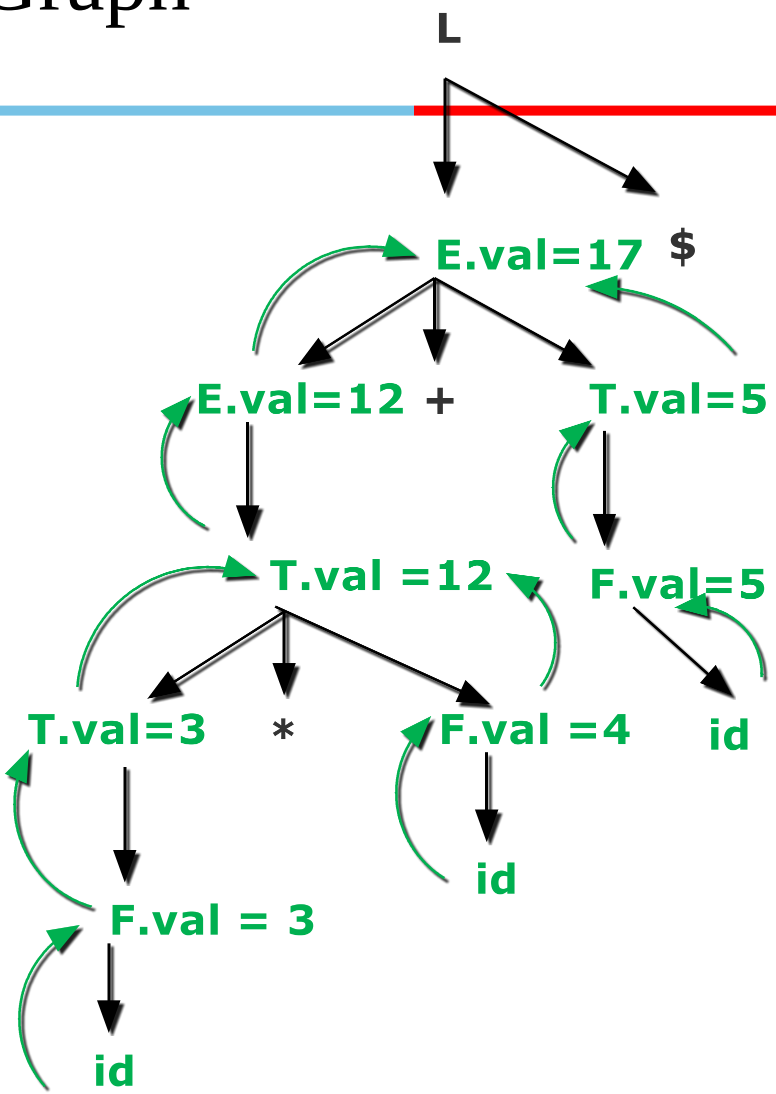
Dependence Graph



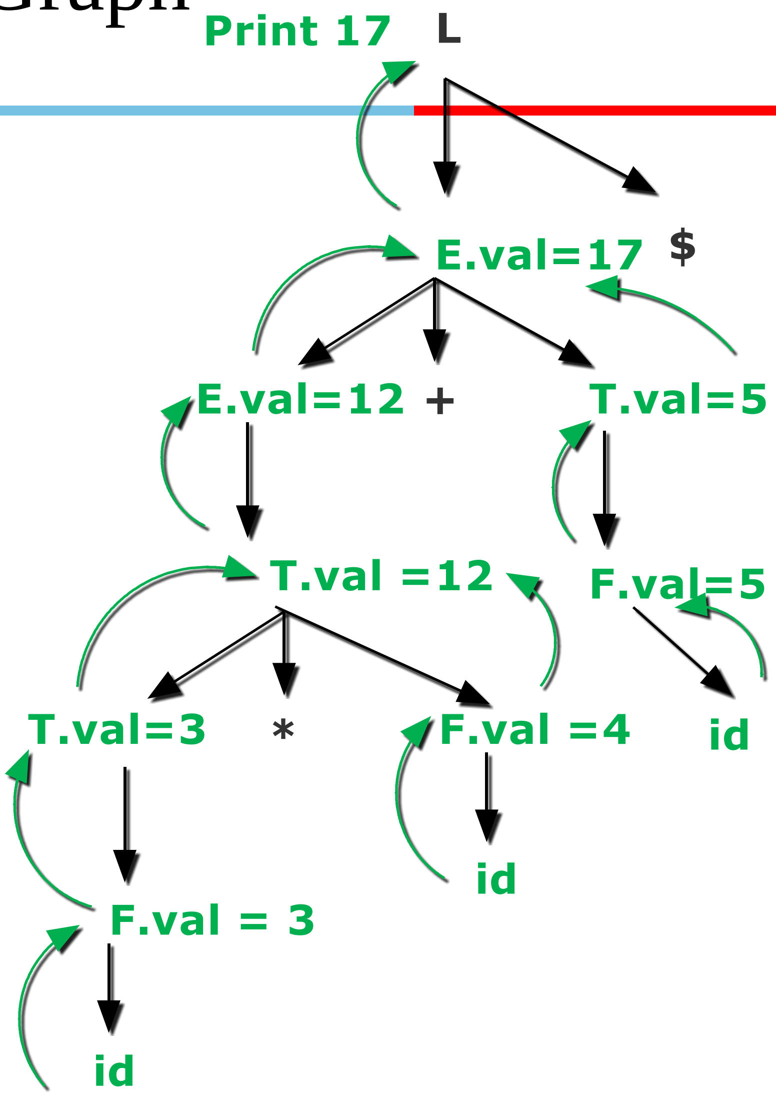
Dependence Graph



Dependence Graph

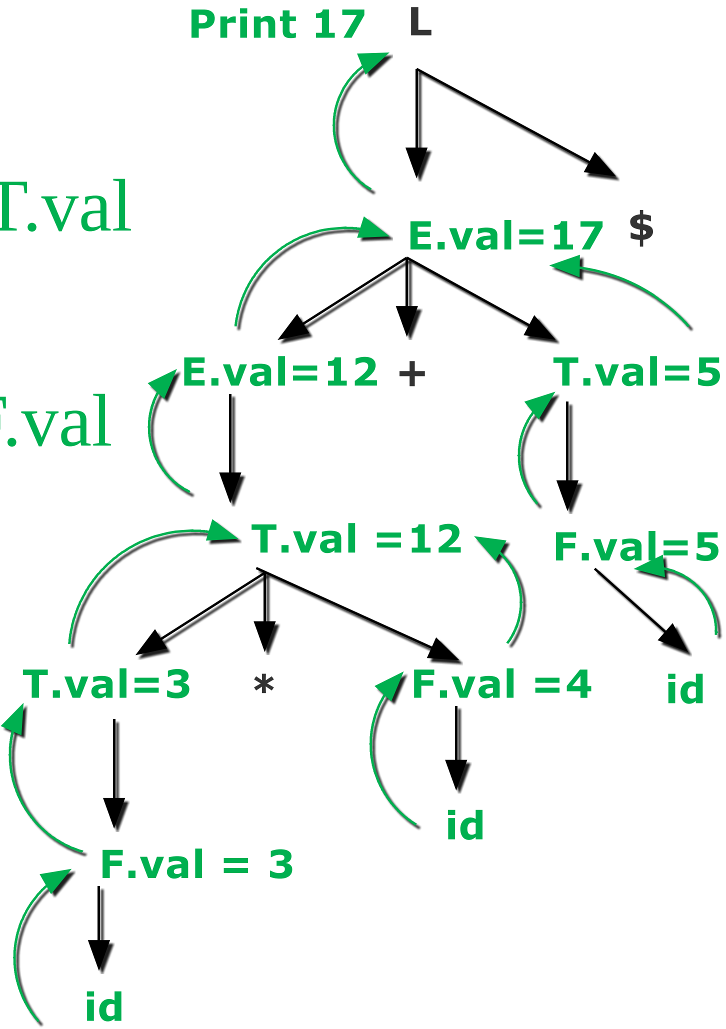


Dependence Graph



Syntax Directed Definitions for a desk calculator program

L → E $	Print (E.val)
E → E + T	E.val = E.val + T.val
E → T	E.val = T.val
T → T * F	T.val = T.val * F.val
T → F	T.val = F.val
F → (E)	F.val = E.val
F → id	F.val = id.lexval


Print 17
L
E.val=17
$
E.val=12
+
T.val=5
T.val =12
F.val=5
T.val=3
*
F.val =4
id
id
F.val = 3
id