CS536
Abstract Syntax Tree (AST)
and
Syntax Directed Translation (SDT)

A Sahu

CSE, IIT Guwahati

http://jatinga.iitg.ac.in/~asahu/cs536/

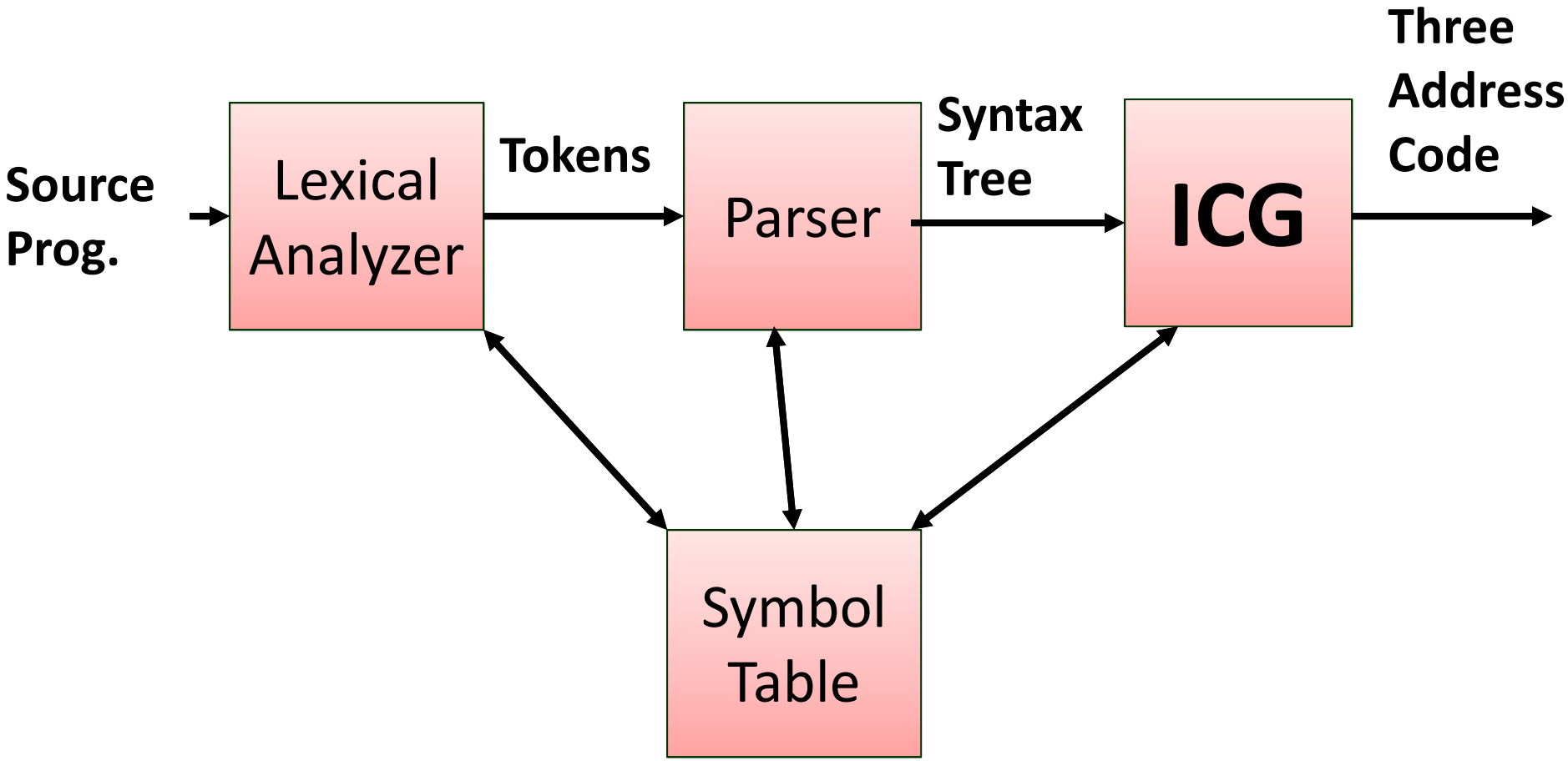
Outline

- Basic of AST
- **Basic of Syntax Directed Translation**
- Intermediate Representation

http://jatinga.iitg.ac.in/~asahu/cs536/

Basic of
Syntax Directed Translation

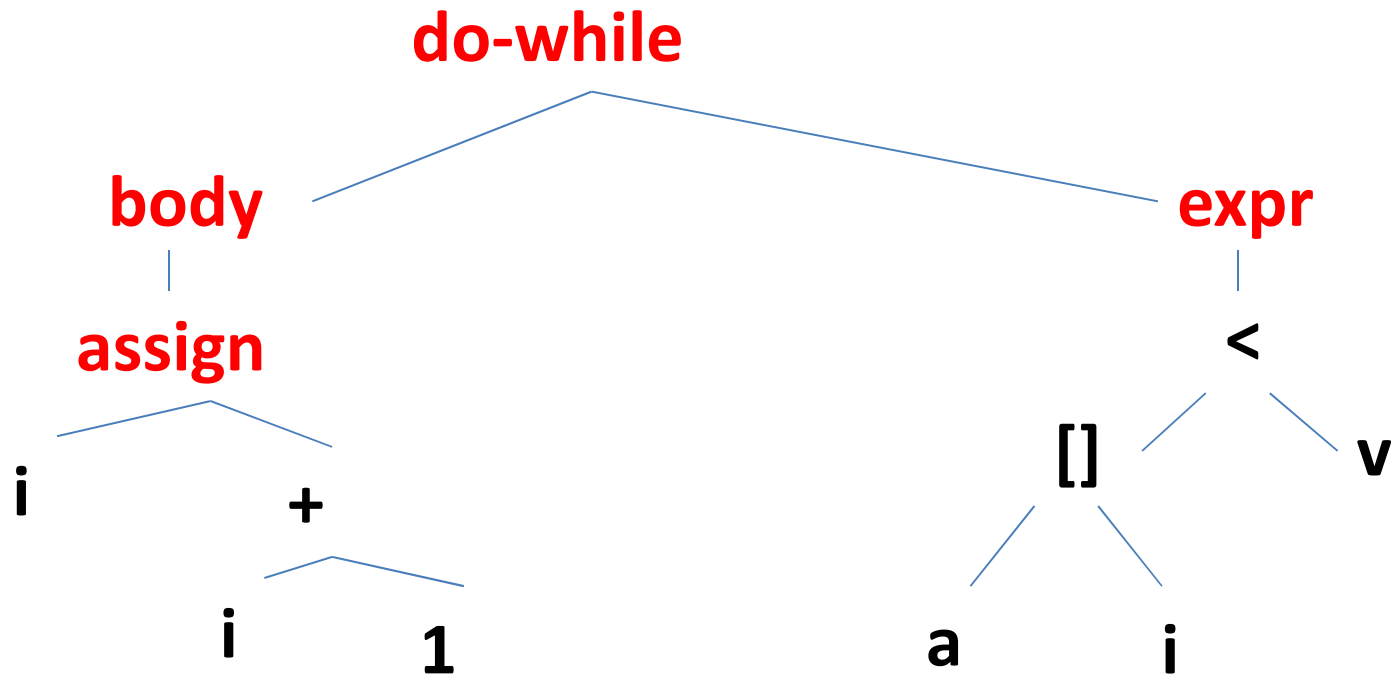
A model of Compiler Front end



ICG: Intermediate Code Generator

Abstract Syntax Tree: AST

- Syntax tree: hierarchical syntactic structure of the source program
- AST for : `do i=i+1; while (a[i]<v);`



Syntax Definition

- A grammar naturally describe the hierarchical structure of the most program

if (expression) statement **else** statement

- Production rule : Can have the form

stmt → **if** (expr) stmt **else** stmt

- In a production: if, else, (,) are terminals
  - The term expr, stmt are non-terminal
  - Can have the form (again)

Definition of Context Free Grammar

- **CFG has four components**
- A set of terminal symbols (referred as token)
  - Elementary Symbols of the Grammar
- S set of non-terminals (NT/syntactic variables)
- A set of production rules
  - Each production of NT called head/left side
  - Arrow and a sequence of T and/or NT called body/right side production
- A designation of one NT as ***Start symbol***

Production Example

Expression : list of digits separated by plus and minus signs

list → list + digit
list→ list – digit
list→digit
digit→0|1|2|3|4|5|6|7|8|9

list → list + digit | list – digit | digit
digit→0|1|2|3|4|5|6|7|8|9

Derivation

- A grammar derives strings by beginning with start symbols
- And repeated replacing a non terminals by body of the production for that non terminals
- The terminal strings can be derived from the start symbol

9-5+2 is list, Can be derived as follows

list → list +2 // list →list + digit

→ list -5 + 2 // list →list - digit

→ 9 – 5 -2 // list → digit

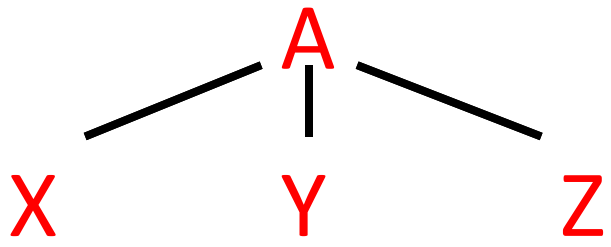
Another Production Example

call → id (optparams)
optparams→ params | ϵ
params → params, param | param

- The term ϵ specifies the empty string
- This analogous/similar to earlier production

Parsing Trees

- A parse tree pictorially shows: How the start symbol of a grammar derives strings in language



1. The root is labeled by start symbol
2. Each leaf is labeled by a terminal or by ϵ
3. Each interior node is labeled by a non-terminals
4. If NT A $X_1, X_2, \ldots, X_n$ are labeled children of A from left to right then there must be production $A \rightarrow X_1, X_2, \ldots, X_n$, where X1, X2, ..Xn are eithe NT or T,
5. If $A \rightarrow \epsilon$, then A may have single child ϵ

Parsing Trees: Properties

- A tree consists of one or more nodes
- **Exactly one root node in a Tree**
  - Root have no-parent, it is top node
  - Other node have exactly one parent
- Leaf: node with no children
- N is parent of M, M is child of N, Children of one node is Siblings, Ordered from left to right
- Descendent (self, child*), Ancestor (self, parent*)

Parse Tree Vs Syntax Tree

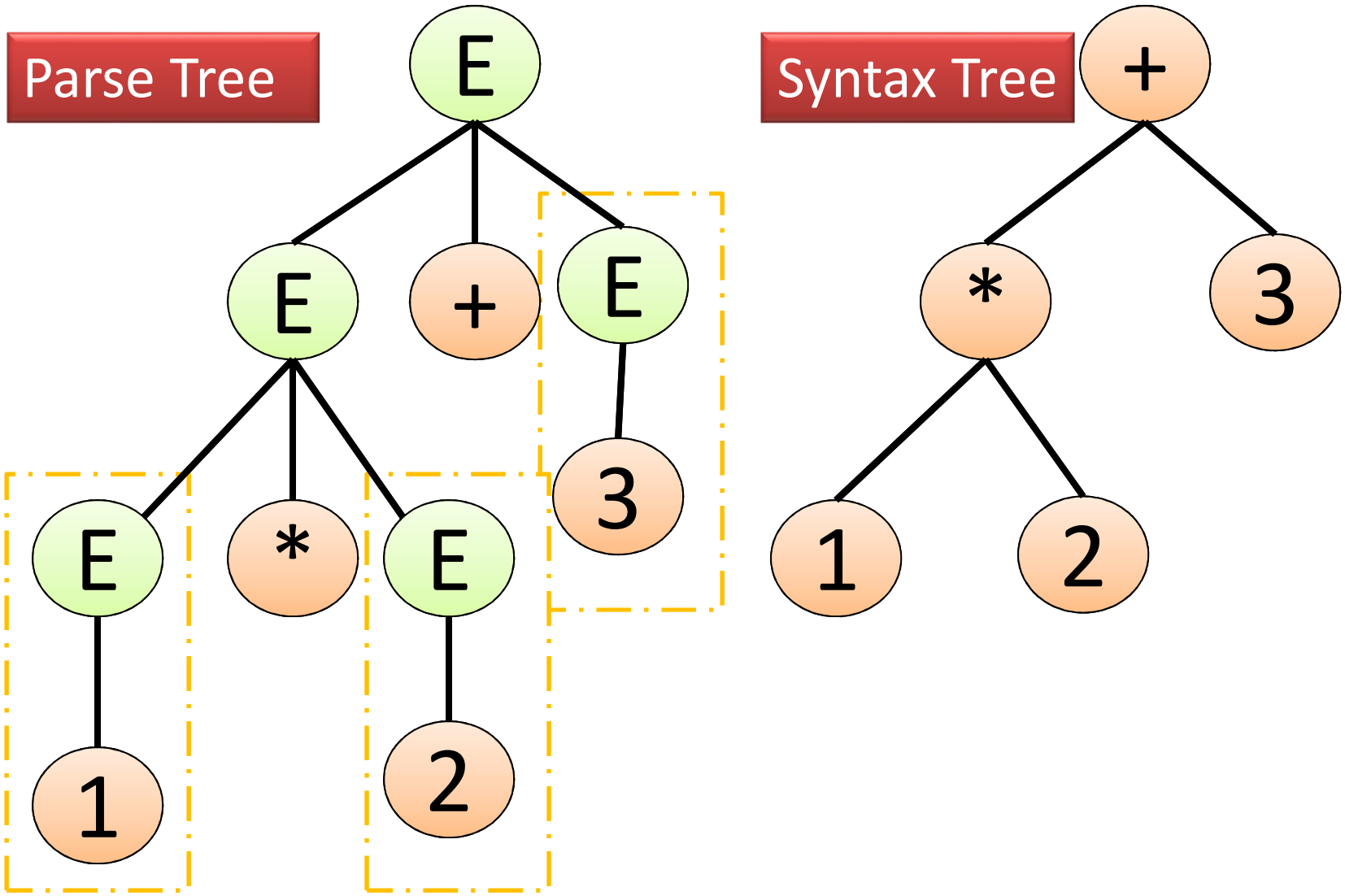
- Parse tree is a hierarchical structure that
  - Defines the derivation of the grammar to yield input strings using start symbol
- Syntax tree: Displays the syntactic structure of a program
  - While ignoring inappropriate analysis present in a parse tree
- Syntax tree is nothing more than a condensed form of the parse tree

Parse Tree Vs Syntax Tree : 1*2+3

- Parse Tree : contain operators & operands at any node of the tree, i.e., either interior node or leaf node.
- Parse contains duplicate or redundant information.
- Parse Tree can be changed to Syntax Tree by the elimination of redundancy, by Compaction

- Syntax contains operands at leaf node & operators as interior nodes of Tree.
- ST do not contains duplicate info.
- Syntax Tree cannot be changed to Parse Tree.

Parse Tree Vs Syntax Tree : 1*2+3

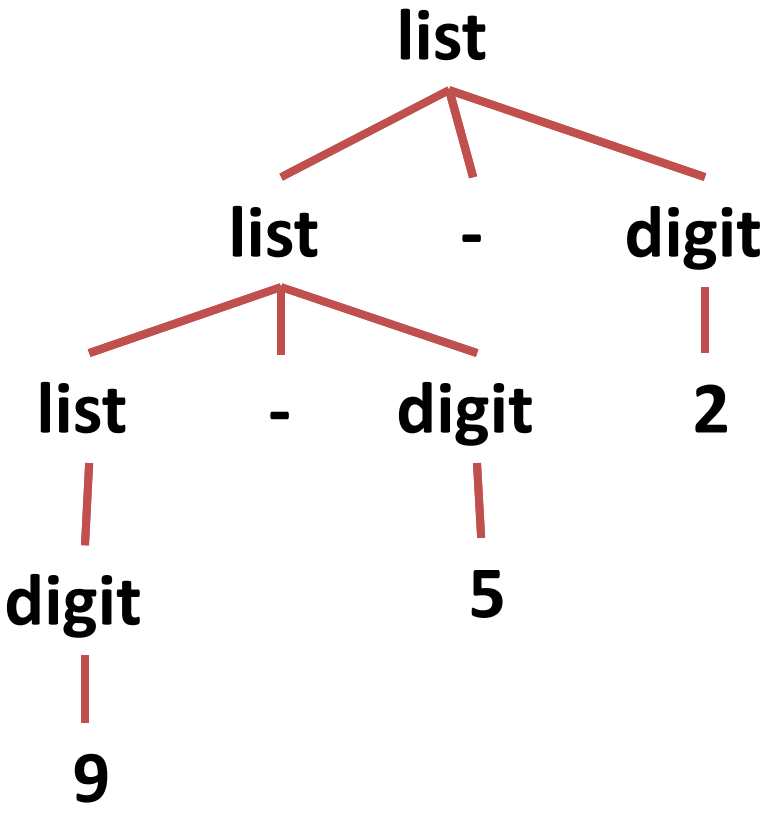


Associativity of Operators

- Equivalence : when ops are same in expr
  - 9+5+2 → (9+5)+2, 9-5-2 → (9-5)-2
  - 9*5*2 → (9*5)*2, 9/5/2 → (9/5)/2
- Left asso: add, sib, mul, div
- Right asso: a=b=c → a= (b=c)

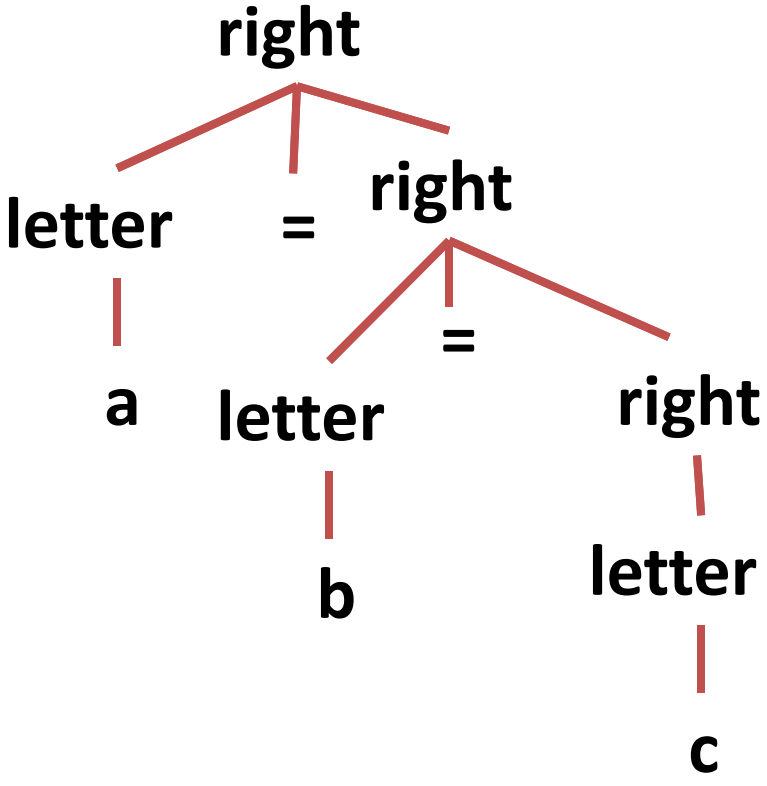
right → letter = right | letter
letter→ a| b| ..| z

Associativity of Operators



Left Associativity

9-5-2



Right Associativity

a=b=c

Precedence of Operators

- Precedence : when ops are different in expr
  - 9+5*2 → *(9+5)*2* or **9+ (5*2)** //

- Left asso: add, sib, mul, div
- Operators on the same line have same asso and precedence

Left-associative: + -
Left-associative: * /

Precedence of Operators

- Consider * and / have higher precedence than + and –
- We create two non-terminal *expr* and *term*
- Non-terminal *factor for generating basic unit of expression*
- The basic unit in expressions are *digit* and *()*

expr → expr+term | expr-term | term
*term → term*factor | term/factor | factor*
factor → digit | (expr)

A Grammar for a subset of C++ Statement

stmt → **id**=*expression*

| **if** (*expression*) *stmt*

| **if** (*expression*) *stmt* **else** *stmt*

| **while** (*expression*) *stmt*

|**do** *stmt* **while** (*expression*)

| {*stmts*}

stmts →*stmts, stmt | ϵ*

Syntax Directed Translation

- **SDT: Attaching rules/Program fragment to the Production rule of Grammar**
- Example production

$$expr \rightarrow expr_1 + term$$

  - ***expr*** is sum of two sub-expressions $expr_1$ and ***term***
- We can **translate** this above production by

translate ***expr1***;
translate ***term***;
handle ***+***;

Syntax-Directed Translation

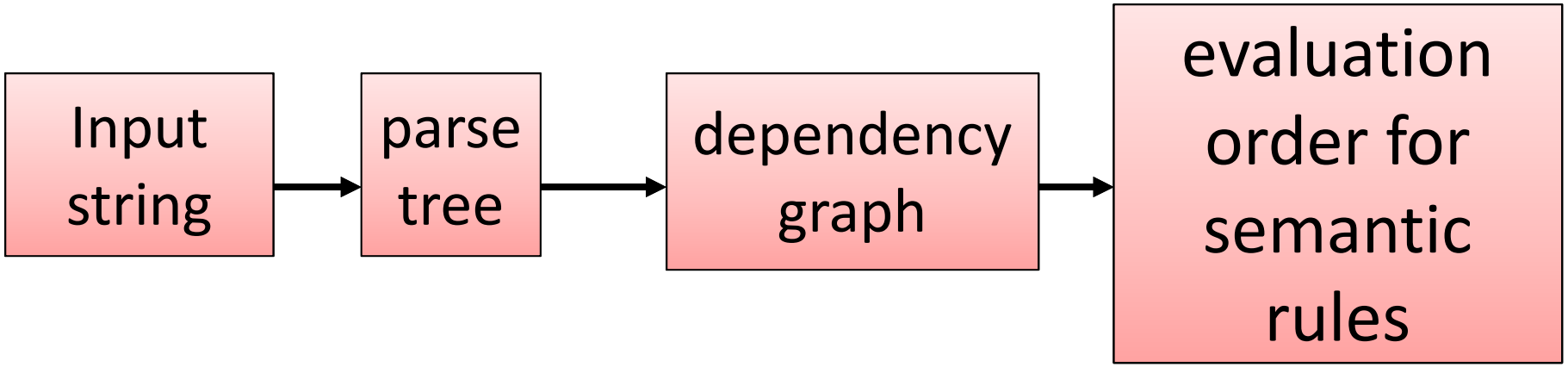
- We associate information with the programming language constructs by attaching attributes to grammar symbols.
- Values of these attributes are evaluated by the **semantic rules** associated with the production rules.
- Evaluation of these semantic rules:
  - may generate intermediate codes
  - may put information into the symbol table
  - may perform type checking
  - may issue error messages
  - may perform some other activities
  - in fact, they may perform almost any activities.
- An attribute may hold almost any thing.
  - a string, a number, a memory location, a complex record.

Syntax-Directed Definitions and Translation Schemes

- When we associate semantic rules with productions, we use two notations
- **Syntax-Directed Definitions:**
  - give high-level specifications for translations
  - hide many implementation details such as order of evaluation of semantic actions.
  - We associate a production rule with a set of semantic actions, and we do not say when they will be evaluated.
- **Translation Schemes:**
  - indicate the order of evaluation of semantic actions associated with a production rule.
  - In other words, translation schemes give a little bit information about implementation details.

Syntax-Directed Translation

- Conceptually with both the syntax directed translation and translation scheme we
  - Parse the input token stream
  - Build the parse tree
  - Traverse the tree to evaluate the semantic rules at the parse tree nodes.



Conceptual view of syntax directed translation

Syntax-Directed Definitions

- Generalization of a context-free grammar in which:
- Each grammar symbol is associated with a set of attributes.
- This set of attributes for a grammar symbol is partitioned into two subsets called
  - **synthesized** attributes and
  - **inherited** attributes of that grammar symbol.
- Each production rule is associated with a set of semantic rules.

Syntax-Directed Definitions

- The value of an attribute at a parse tree node
  - is defined by the semantic rule associated with a production at that node.
- Value of a **synthesized attribute** at a node is
  - computed from the values of attributes at the children in that node of the parse tree
- Value of an **inherited attribute** at a node is
  - computed from the values of attributes at the siblings and parent of that node of the parse tree

Syntax-Directed Definitions

Examples: Synthesized attribute :

$$E \rightarrow E1+E2 \quad \{ E.val = E1.val + E2.val \}$$

- *Semantic rules* set up dependencies between attributes which can be represented by a ***dependency graph***.
- This *dependency graph* determines the evaluation order of these semantic rules.
- Evaluation of a semantic rule defines the value of an attribute.
  - But a semantic rule may also have some side effects such as printing a value.

Annotated Parse Tree

1. A parse tree shows the values of attributes at each node is called an **annotated parse tree**.
2. Values of Attributes in nodes of annotated parse-tree are either,
   - initialized to constant values or by the lexical analyzer.
   - determined by the semantic-rules.
3. The process of computing the attributes values at the nodes is called **annotating** (or **decorating**) of the parse tree.
4. Of course, the order of these computations depends on the dependency graph induced by the semantic rules.

Syntax Directed Translation

- **Attributes:** Quantity associated with a programming construct. Examples
  - Data type/Size of an expression
  - Number of instruction in generated code
  - Location of first instruction
  - Constructs: symbols, terminals, non-terminals
- **SD Translation Scheme**
  - **Notion of attaching program fragment to the Production**
  - The program fragment are executed when the production is used during syntax analysis

Thanks