CS536
Syntax Directed Translation (SDT)
&
Symbol Table

A Sahu

CSE, IIT Guwahati

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

Outline

- **Basic of Syntax Directed Translation**
- Symbol Table
- Intermediate Representation

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

Basic of
Syntax Directed Translation

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.

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

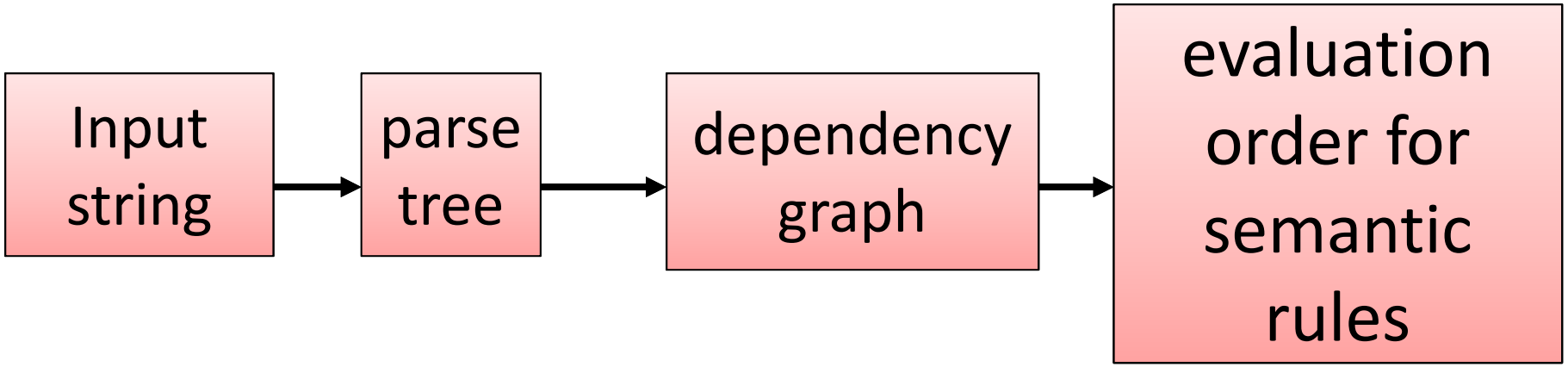
- We associate information with the programming language constructs by attaching attributes to grammar symbols.
- Values of these attributes are evaluated by the **semantic rules** associated with the production rules.
- Evaluation of these semantic rules:
  - may generate intermediate codes
  - may 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

SDT Example: Postfix Notation

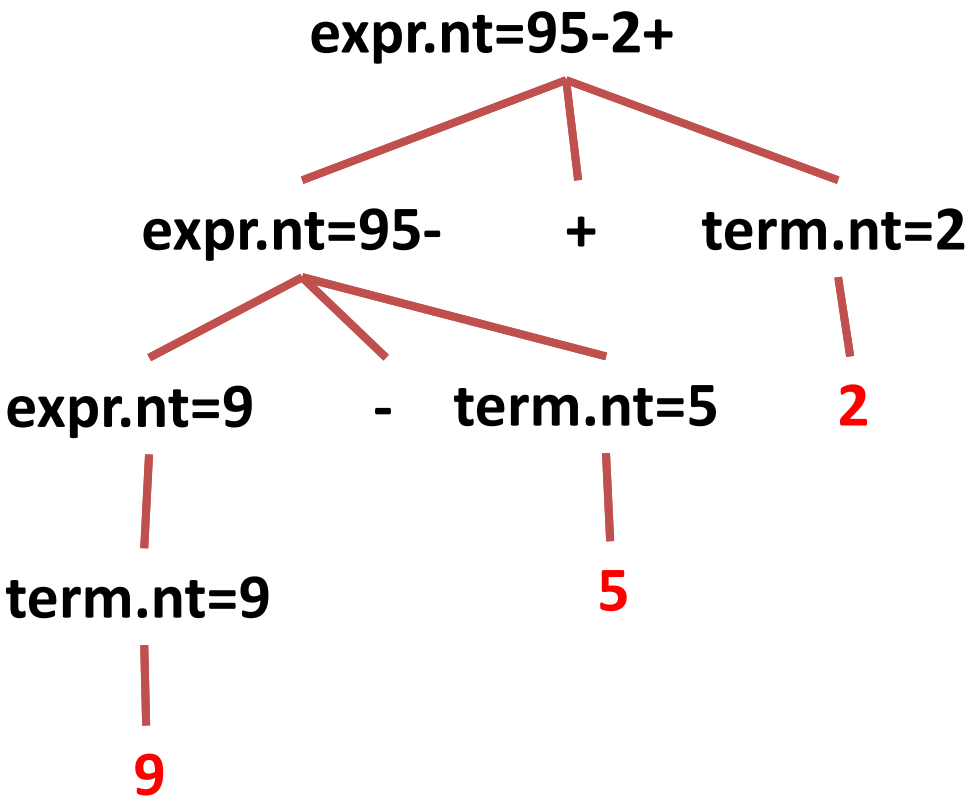
- Postfix notation for an expr *E*
  - If *E* is variable |const, P(*E*)=*E*
  - If *E* is of form E_1 ***op*** E_2 the P(E) = E_1' E_2' ***op***, where E_1'=P(E_1) and E_2' =P(E_2)
  - If E in the form of parenthesized (E1), P(E) = P((E$_1$))=P(E$_1$)
- Example: P(***(9-5)+2***)=95-2+
- No parentheses are needed in PN
  - **Position and arity (number of arguments) of the operator permits only one decoding of the PN**

Synthesized Attributes

- Idea of associating quantities with construct
- Value and types with expressions
- Associate attributes with non-terminals and terminals
- After that production rule is attached.
- A **Syntax directed definition** associates
  - With each grammar symbol, a set of attributes
  - With each production, a set of semantic rule for computing the value of attributes associated with symbols

Synthesized Attributes

- A node in Parse tree is label by grammar symbol X
- X.a is value of attribute a of X at that node
- A parse tree showing the attribute values at each node is called annotated parse tree
- Suppose ***nt*** is associated with all non-terminal **term** and ***expr***.



Attribute values at nodes in parse tee

Synthesized Attributes

- A Attribute is said to be synthesized
  - if its value at a parse tree node N is determined from
  - the attribute values at the children of N and N it self
- Synthesized attributed have the desirable properties that
  - Can be evaluated during a single bottom-up traversal of the parse tree

SD definition example: postfix notation

Production	Semantic Rules
expr→expr$_1$+term	expr.nt=expr$_1$.nt | term.nt | '+'
expr→expr1+term	expr.nt=expr$_1$.nt | term.nt | '-'
expr→term	expr.nt=term.nt
term→0|1|2|..|9	term.nt='0'|'1'|..|'9'

- **SD Definition is simple**
- **If** the string representing translation of NT at the head of each production is
  - The concatenation of translation of the NT in the production body in the same order
  - with some optional additional strings interleaved

Top Down Parsing

- This is example of Grammar to generate subset of statement in C/Java

stmt → *expr*

| **if** (*expr*) stmt

| **for**(*optexpr;optexpr; optexpr*) *stmt*

| **others**

optexpr → *expr* | ϵ

Top Down Parsing

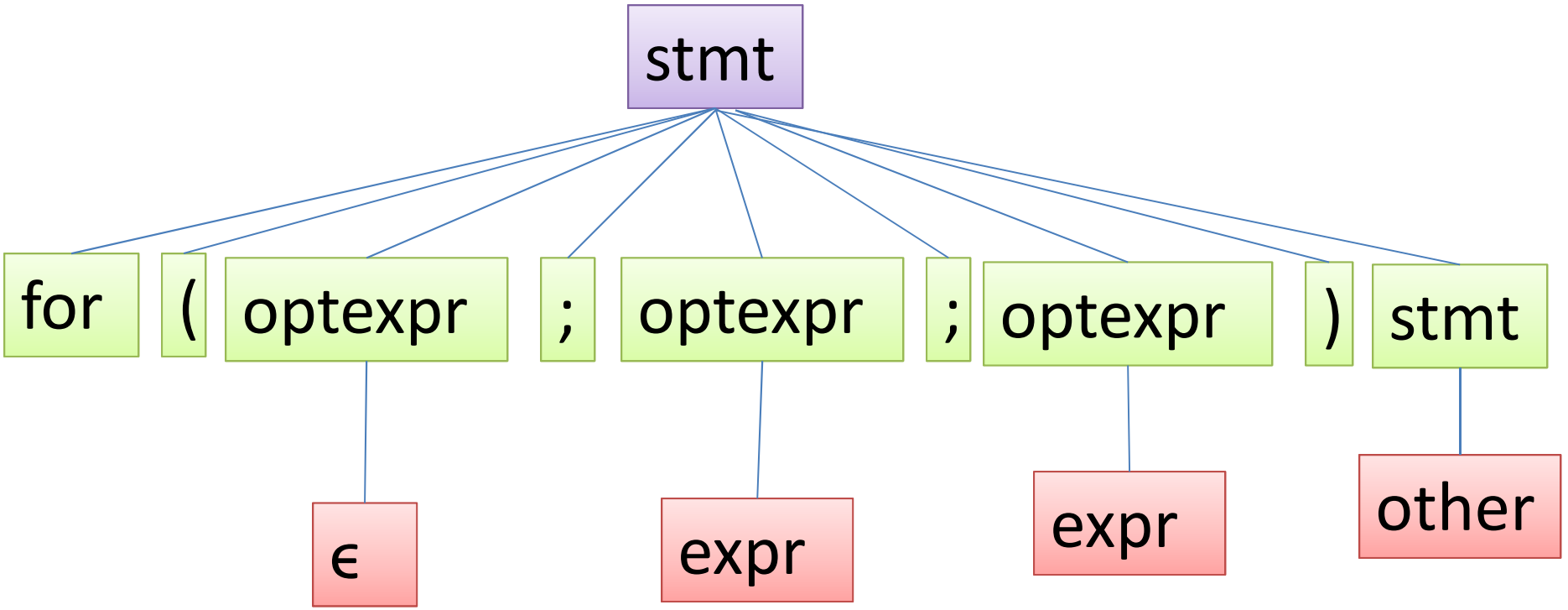
- Top Down Parsing can be done by starting with the root, labeled with NT stmt and repeatedly doing
  - At node N, labeled with non terminal A, select one production for A and children at N for the symbols
  - Find the next node at which a subtree is to constructed, typically the leftmost unexpanded NT of the tree
- The above steps during a single left-to-right scan of the input
- The current symbol being scan is ***lookahead.***

Top Down Parsing: example

- Suppose input string is

```
for ( ; expr; expr ) other
```

- Tree for this is



Predictive parsing

- Recursive - descendent parsing (RDP) is a top down parsing of syntax analysis
  - Also called the predictive parsing
- One procedure associated with
  - each non-terminal of a grammar
- In RDP, lookahead symbol
  - unambiguously determined the flow of control through the procedure body for each non-terminals
- The sequence of procedure calls during the analysis of an input string
  - Implicitly define the parse tree for the input

Code for predictive parser

```
void stmt() {
   switch ( lookahead ) {
case expr        :  MC(expr); MC(' ; '); break;
case if          : MC(if); MC(' (‘); MC (expr); MC(') '); stmt (); break;
case for         : MC (for); MC (' (') ; optexpr (); MC(' ; ‘);
                   optexpr(); MC(' ; '); optexpr(); MC(') '); stmt (); break;
case other       :  MC (other) ; break;
default          : report ("syntax error");
}
```

```
void optexpro { if ( lookahead == expr ) MC(expr); }
```

```
void MC(termina1 t) {
        if (Lookahead == t ) lookahead = nextTerminal;
        else report ("syntax error”);
}
```

Thanks