Course: Compiler Construction

Week 8: Compiler Design Phases – Semantic Analysis

Lecturer: Martha Gichuki

Course Description

- The course begins with a coverage of basic Principles in Compiler Design with an introduction to Formal programming language translation and compiler design concepts; compilers, translators and interpreters.
- A coverage of Language theory, Parsing Context-Free Languages (Top - down and bottom - up parsing), translation specifications and machine- independent code optimization will then follow.

Course Description …

- The main phases of compilation i.e. Lexical, Syntax and Semantic analysis will be taught.
- Code generation and Optimization, Symbol table design, Program compilation, Loading and execution will then be covered.
- Compilation techniques, Optimization, Design of a simple complete compiler, will culminate the course.

Learning outcomes Week 8: Compiler Design Phases – Semantic Analysis

At the end of the lecture, you will be able to:

i. Define Semantics of a language and attributes of grammar,
ii. Differentiate Synthesized and Inherited attributes
iii. Describe Syntax Directed Translation

SEMANTIC ANALYSIS

- ✓Semantic analysis checks the source program for **semantic consistency** with the language definition i.e. parsers cannot handle context-sensitive features of programming languages[1]
- ✓Examples of **static semantics** of programming languages that can be checked by the semantic analyzer are: -
  i. Variable cannot be used without having been defined
  ii. Same variable /function/class/method cannot be defined twice
  iii. Types match on both sides of assignments
  iv. Parameter types and number match in declaration and use

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 33

- ✓ Compilers can only generate code to check **dynamic semantics** of programming languages at runtime such as[2]: -

i. Will an overflow occur during an arithmetic operation?

ii. Will **array limits** be **crossed** during execution?

iii. Will **recursion cross the stack limits?**

iv. Will the **heap memory** be **sufficient?**

2. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 241

- ✓ In general, the plain parse-tree constructed during the syntax analysis is of no use for a compiler, since **it does not carry any information of how to evaluate the tree**[3]
- ✓ For example, **E → E + T** has no semantic rule associated with it, and it cannot help in making any sense out of the production.

3. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 312

Semantics of a language

- ✓ Language Semantics provide meaning to its constructs, e.g. **tokens and syntax structure**.
- ✓ Semantics help **interpret symbols**, their **types**, and **their relations** with each other.
- ✓ Semantic analysis judges whether the syntax structure constructed in the source program **derives any meaning** or not.
- ✓ Example**: int a = "string value ";**

This statement should not issue an error in lexical and syntax analysis phases, as it is lexically and structurally correct, but it should generate a **semantic error** as the type of the assignment differs.

These rules are set by the grammar of the language and evaluated in semantic analysis[4].

4. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 312

Syntax Directed Translation (SDT) [5]

- ✓ Syntax-directed translation is done by attaching rules or program fragments to productions in a grammar.

The following are **two concepts** related to syntax-directed translation:

i. An **attribute** is any quantity associated with a programming construct. Examples of attributes:
   a) data types of expressions,
   b) the number of instructions in the generated code,
   c) or the location of the first instruction in the generated code for a construct.

5. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 303

ii). A **translation scheme** is a notation for attaching program fragments to the productions of a grammar.

- ✓ The program fragments are executed when the production is used during syntax analysis.
- ✓ The combined result of all these fragment executions, in the order induced by the syntax analysis, produces the translation of the program to which this analysis/synthesis process is applied[6].

6. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 324

ATTRIBUTE GRAMMAR

- ✓ Attribute grammar is a special form of Context-Free Grammar(CFG) where some additional information (attributes) provides context-sensitive information.
- ✓ It is a medium to provide semantics to the CFG and helps to specify the **syntax and semantics** of a programming language.
- ✓ Attribute grammar (when viewed as a parse-tree) can pass values or information among the nodes of a tree[7].

7. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 306

- ✓ The idea behind attribute grammar is to **associate attributes** with **each node** in the (abstract) syntax tree[8].
- ✓ **Examples of attributes include:-**

i. Type information

ii. Storage location

iii. Assignable (e.g., expression vs variable)

iv. Value (for constant expressions)

8. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 318

- ✓It uses the notation **X.a**, where **a** is an attribute of node **X**
- ✓**Attributes** are associated with **non-terminals** and **terminals**
- ✓**Rules** are attached to the **productions** of the grammar and they **describe how attributes are computed** at the **parse tree nodes** where the **production** in question is used to **relate a node to its children**[9].

9. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 318

- ✓A syntax-directed definition associates with[10]:-

i. Each grammar symbol,

ii. A set of attributes, and

iii. Each production, a set of **semantic rules** for computing the values of the attributes associated with the symbols appearing in the production.

- ✓A parse tree showing the attribute values at each node is called **an annotated parse tree**

10. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 318

Example:- Consider the following grammar[11]

$$L \rightarrow E$$

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow (E)$$

$$F \rightarrow \mathit{num}$$

11. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 326

A table showing attributes and rules that calculate the value of an expression[12]

$L \rightarrow E$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow T * F$
$T \rightarrow F$
$F \rightarrow (E)$
$F \rightarrow \mathbf{num}$

Production	Semantic Rules
$L \rightarrow E$	$L.val = E.val$
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow \mathbf{num}$	$F.val = num.lexval$

Note: Subscripts enable us to distinguish ***different instances of the same symbol in a rule***

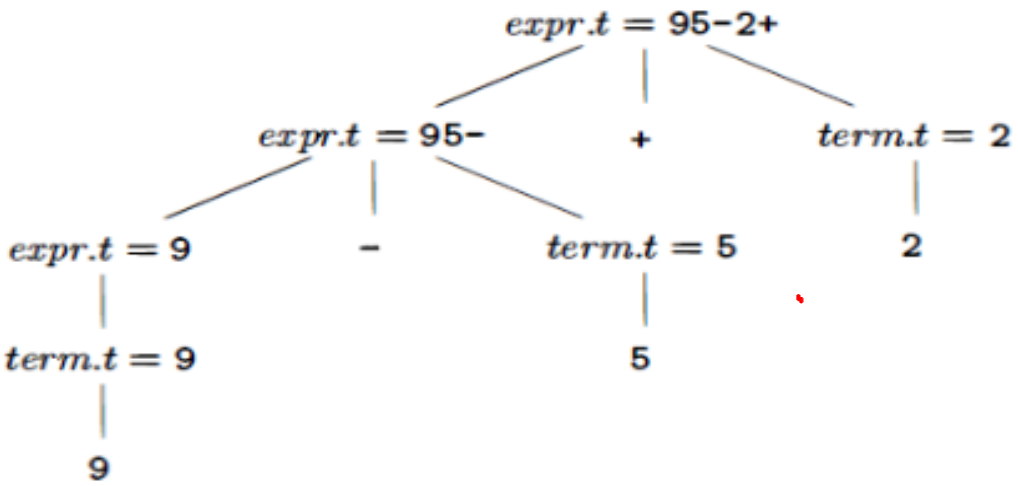
12. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 326

Code Translation

- ✓ Syntax-directed definitions can be used to translate code e.g. translating expressions to **post-fix notation**

Example: An annotated parse tree for **9-5+2** with an attribute ***t*** associated with the non-terminals ***expr*** and ***term***.

The value **95-2+** of the **attribute at the root** is the **postfix notation** for **9-5+2**[13].



13. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 325

- ✓ The annotated parse tree is based on the syntax directed definition below for translating expressions consisting of **digits separated by plus or minus signs into postfix notation.**
- ✓ Each **non-terminal** has a **string-valued attribute t** that represents the **postfix notation** for the **expression** generated by that **non-terminal** in a parse tree[14].

14. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 324

The symbol || in the semantic rule is the operator for **string concatenation**[15].

Production	Semantic Rules
$expr \rightarrow expr_1 + term$	$expr.t = expr_1.t \,||\, term.t \,||\, '+'$
$expr \rightarrow expr_1 - term$	$expr.t = expr_1.t \,||\, term.t \,||\, '-'$
$expr \rightarrow term$	$expr.t = term.t$
$term \rightarrow 0$	$term.t = '0'$
$term \rightarrow 1$	$term.t = '1'$
…	…
$term \rightarrow 9$	$term.t = '9'$

- ✓ Postfix form of a digit is the digit itself; e.g. the semantic rule associated with the production: **term → 9** defines **term.*t*** to be **9** itself whenever this production is used at a node in a parse tree.
- ✓ Other digits are translated in a similar manner. Applying the production **expr → term** means the value of **term.t** becomes the value of **expr.t**

15. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 324

Types of Attributes[16]

- ✓ Based on the **way the attributes get their values,** they can broadly be divided into two categories namely[16]:

i). Synthesized attributes

ii). Inherited attributes

16. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 303

i). Synthesized Attributes[17]

- ✓ These attributes get values from the **attribute values of their child nodes** i.e. attribute value for the **LHS** non-terminal is computed from the **attribute values of the symbols at the RHS of the rule**.
- ✓ **Example:** Consider the following production: S → ABC

If S is taking values from its child nodes (A, B, C), then it is said to be a synthesized attribute, as the values of ABC are synthesized to S.

17. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 304

- ✓ Synthesized attributes never take values from their parent nodes or any sibling nodes.
- ✓ Terminals can have synthesized attributes, computed by the lexer (e.g., id. lexeme), but no inherited attributes[18].

18. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 303

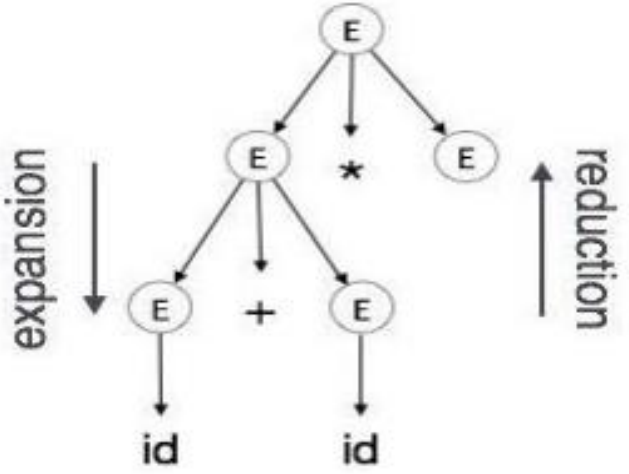
ii). Inherited Attributes[19]

- ✓ Unlike synthesized attributes, inherited attributes **can take values from parent and/or siblings**
- ✓ Attribute value of a RHS non-terminal is **computed from the attribute values of the LHS** non-terminal and some other RHS non-terminals.
- ✓ E.g. from **S → ABC;**
  - **A** can get values from **S, B, C**.
  - **B** can take values from **S, A,** & **C.**
  - **C** can take values from **S, A,** & **B**.

19. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 304

Expansion vs Reduction[20]

- ✓**Expansion** happens when a **non-terminal** is **expanded to terminals** as per a grammatical rule.
- ✓**Reduction:** happens when a **terminal is reduced** to its corresponding **non-terminal** according to grammar rules.
- ✓Syntax trees are parsed top-down and left to right and whenever reduction occurs, we apply its corresponding semantic rules (actions).



20. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 348

Generally, an attribute grammar is a way of relating strings with "**meanings**"

- ✓Since this relation is syntax-directed, we associate each CFG rule with a semantics (rules to build an abstract syntax tree)
- ✓In other words, attribute grammars are a method to decorate or annotate the parse tree
- ✓The semantic analyzer receives Abstract Syntax Tree (AST) from the syntax analysis and then attaches attribute information to the AST.
- ✓This AST are called Attributed AST[21].

21. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 318

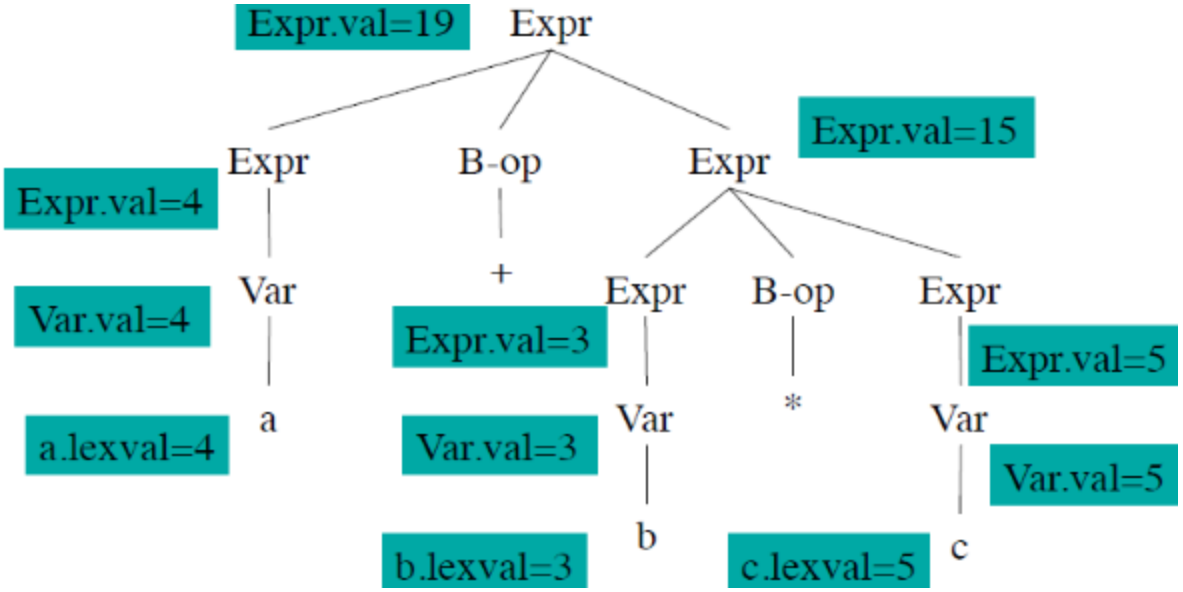
✓Attributes are **two tuple value, <attribute name, attribute value>** as shown below:

int value = 5;

<type, "integer">

<presentvalue, "5">

i.e. for **every production**, you need to **attach a semantic rule**. Example[22]:



22. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 312

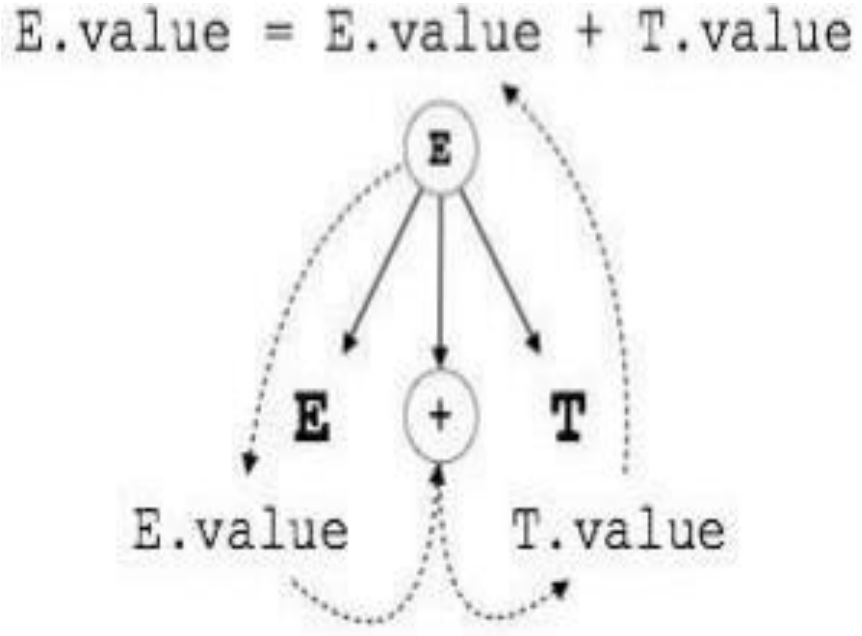
Flow of Attributes in Expr[23]

- ✓ Consider the flow of the attributes in the Expr syntax-directed definition
- ✓ The LHS attribute is computed using the RHS attributes
- ✓ Purely **bottom-up**: compute attribute values of all children (RHS) in the parse tree and then use them to compute the attribute value of the parent (LHS)

23. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 310

A. S-attributed SDT

- ✓If an SDT uses **only synthesized attributes, it is called S-attributed SDT**.
- ✓Such a grammar plus semantic actions is called an S-attributed definition i.e. a grammar with semantic actions (or syntax-directed definition) can choose to use only synthesized attributes for example[24]:-



24. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 312

- ✓As shown above, attributes in S-attributed SDTs are evaluated in **bottom-up parsing**, as the values of the parent nodes depend upon the values of the child nodes.

B. L-attributed SDT

- ✓This form of SDT **uses both synthesized and inherited attributes** with **restriction of not taking values from right siblings**.
- ✓In L-attributed SDTs, **a non-terminal can get values from its parent, child, and sibling nodes**[25].

25. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 346

L-attributed SDT …

- ✓Consider the production: S → ABC
  - S can take values from A, B, and C (synthesized).
  - A can take values from S only.
  - B can take values from S and A.
  - C can get values from S, A, and B.
- ✓Attributes in **L-attributed SDTs** are evaluated by **depth-first** and **left-to- right** parsing manner.
- ✓**Note:** If a definition is S**-attributed**, then it is also **L-attributed**, since L-attributed definition **encloses** S- attributed definitions[26].

26. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison- Wesley Pub Co, ISBN: 0201100886 (2007) page 343

Two important classes of SDTs:

I. **LR parser**, syntax directed definition is **S-attributed**

✓To implement S-attributed definitions in LR parsing, we **execute action on reduce, all necessary attributes have to be on the stack**

II. **LL parser**, syntax directed definition is **L-attributed**

✓To implement L-attributed definitions in LL parsing, we use **an additional action record for storing synthesized and inherited attributes on the parse stack**[27]

27. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 331

Applications of Syntax-Directed Definitions

✓SDD can be used at several places during compilation:

i. Building the syntax tree from the parse tree

ii. Various static semantic checking (type, scope, etc.)

iii. Code generation

iv. Building an interpreter[28]

28. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; Addison-Wesley Pub Co, ISBN: 0201100886 (2007) page 338

Content Covered in Week 8: Compiler Design Phases – Semantic Analysis

At the end of the lecture, we were able to:

(i) Define Semantics of a language and attributes of grammar

(ii)Differentiate Synthesized and Inherited attributes

(iii) Describe Syntax Directed Translation

Course Text Books

1. Compilers: Principles, Techniques and Tools, Aho, Lam, Sethi, and Ullman; *Addison- Wesley Pub Co,* ISBN: 0201100886 (2007))
2. Compiler Construction: Principles and practices; Kenneth C. Louden; *Cengage Learning*; 1st edition, ISBN-10 : 0534939724 (1997)
3. Basics of Compiler Design: Torben Mogensen; *DIKU University of Copenhagen Universitetsparken 1 DK-2100 Copenhagen DENMARK* (2007)
4. Engineering a Compiler: 2nd edition; Keith D. Cooper and Linda Torczon; *Morgan Kaufmann Publishers* ISBN: 978-0-12-088478-0 (2003)
5. Compiler Design: Santanu Chattopadhyay; *PHI Learning Publishers*, ISBN 812032725X, 9788120327252 (2005)