COMPILER (CSE4120)

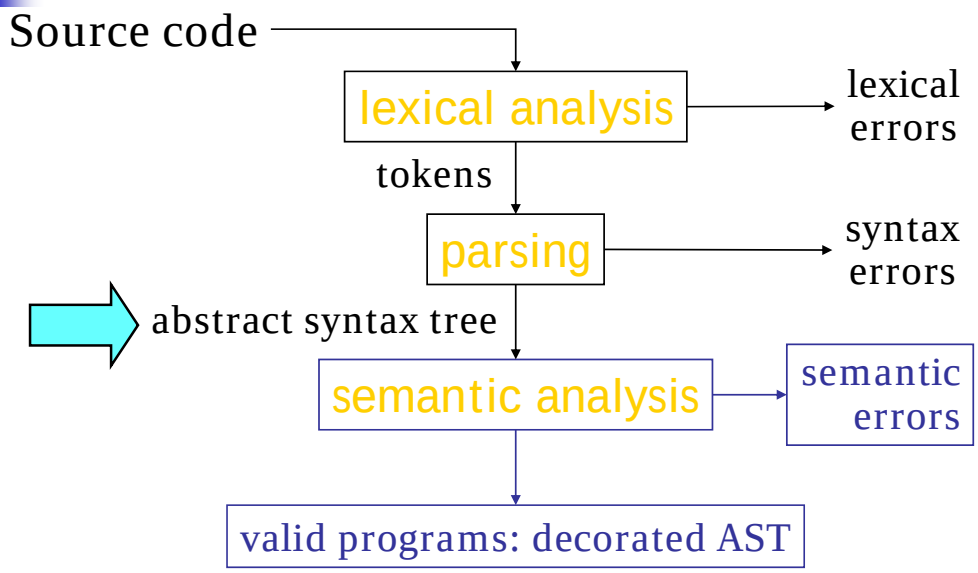
(Lecture 08: Semantic Analysis 1)

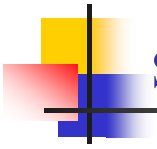
Sungwon Jung, Ph.D.

Mobile Computing and Data Engineering Lab.
Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea
Tel: +82-2-705-8930
Email : jungsung@sogang.ac.kr

Dept. of Computer Science at Sogang University

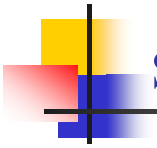
Semantic Analysis





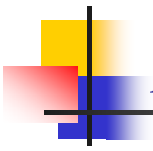
Semantic Analysis

- Computes additional information needed for compilation once the syntactic structure of a program is known
  - Involves computing information that is beyond the capabilities of CFG and standard parsing algorithms
    - Building a symbol table to keep track of the meanings of names in declarations
    - Performing type inference and type checking on expressions and statements to determine their correctness within the type rules of the language



Semantic Analysis

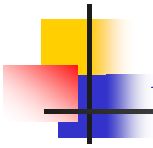
- Description of semantic analysis
  - Identify **attributes** of language entities that must be computed
  - Write **attribute equations** or **semantic rules** that express how the computation of attributes is related to the grammar rules of the language
- Attribute grammar consists of a set of attributes and equations
  - Most useful for languages that obey the principle of **syntax-directed semantics**
    - Syntax-directed semantics
      - The semantic content of a program is closely related to its syntax



Attributes and Attribute Grammars

- Attributes: any property of a programming language construct
  - Examples:
    - The data type of a variable
    - The value of an expression
    - The location of a variable in memory
    - The object code of a procedure
    - The number of significant digits in a number
  - Binding of the attribute
    - the process of computing an attributes and associating its computed value with the language construct
      - Binding time: Refer to the time during the compilation/execution process when the binding of an attribute occurs
        - Binding times of different attributes vary: **Static** vs. **Dynamic** attributes

Dept. of Computer Science at Sogang University



Attributes and Attribute Grammars

- Examples of attributes for an identifier
  - name: character string, obtained from lexical analyzer
  - scope
  - type:
    - integer
    - array
      - number of dimensions
      - upper and lower bounds for each dimension
      - type of elements
    - record
      - name and type of each field
    - function
      - number and types of parameters
      - type of returned value
      - entry point in memory and size of stack frame

Syntax-Directed Semantics

- Attribute grammar for a simple desk calculator

Production	Semantic Rules
$L \rightarrow E$ **n**	$print(E.val)$
$E \rightarrow E_1 + T$	$E.val := E_1.val + T.val$
$E \rightarrow T$	$E.val := T.val$
$T \rightarrow T_1 * F$	$T.val := T_1.val \times F.val$
$T \rightarrow F$	$T.val := F.val$
$F \rightarrow (E)$	$F.val := E.val$
$F \rightarrow$ **digit**	$F.val :=$ **digit**$.lexval$

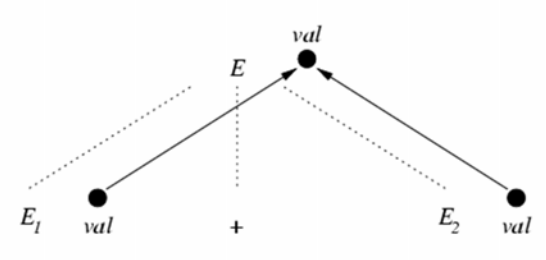
Dept. of Computer Science at Sogang University

Syntax-Directed Semantics

- Dependency graphs
  - Definition:
    - a *directed graph* representing the interdependencies among the attributes at the nodes in a parse tree
  - An attributes *b* depends on attribute *c*
    - The semantic rule *b* must be evaluated after the semantic rule *c*
  - Example:
    - Suppose A.*a* = *f*(X.*x*, Y.*y*) is a semantic rule for the production A→XY
    - An attribute A.*a* depends on the attribute X.*x* and Y.*y*

Syntax-Directed Semantics

- Parse tree and dependency graph
  - Example:
    - Production: $E \rightarrow E_1 + E_2$
    - Semantic rule: $E.val = E_1.val + E_2.val$

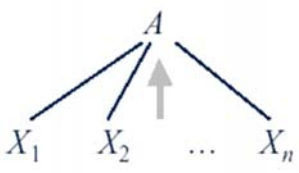


    - $E.val$ depends on $E_1.val$ and $E_2.val$

Dept. of Computer Science at Sogang University

Synthesized Attributes

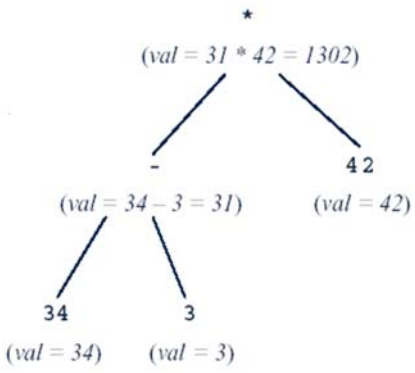
- Definition
  - An attribute is synthesized if all of its dependencies point from child to parent in the parse tree
- Attribute ***a*** is synthesized if
  - A grammar rule $A \rightarrow X_1 X_2 \ldots X_n$
  - $A.a = f(X_1.a_1, \ldots, X_1.a_k \ldots X_n.a_1, \ldots, X_n.a_k)$
  - An attribute grammar in which all the attributes are synthesized is called an S-attributed grammar



Synthesized Attributes

- The attribute values of an S-attributed grammar can be computed by a single bottom-up, or postorder traversal of the tree.

procedure *PostEval* (*T* : *treenode*)**;**
begin
 for *each child C of T* **do**
  *PostEval(C)***;**
 *compute all synthesized attributes of T***;**
end;



Dept. of Computer Science at Sogang University

Structure of a Syntax Tree

- Examples

Grammar Rule	Semantic Rules
$exp_1 \rightarrow exp_2$ + *term*	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2$ - *term*	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2$ * *factor*	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
factor → (*exp*)	$factor.val = exp.val$
factor → **number**	$factor.val =$ **number**$.val$

C definitions

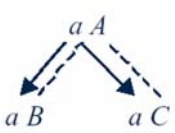
```
typedef enum {Plus,Minus,Times} OpKind;
typedef enum {OpKind,ConstKind} ExpKind;
typedef struct streenode
      { ExpKind kind;
        OpKind op;
        struct streenode *lchild,*rchild;
        int val;
      } STreeNode;
typedef STreeNode *SyntaxTree;
```

- C code for the postorder attribute evaluator

```
void postEval(SyntaxTree t)
{ int temp;
  if (t->kind = OpKind)
  { postEval(t->lchild);
    postEval(t->rchild);
    switch (t->op)
    { case Plus:
        t->val = t->lchild->val + t->rchild->val;
        break;
      case Minus:
        t->val = t->lchild->val - t->rchild->val;
        break;
      case Times:
        t->val = t->lchild->val * t->rchild->val;
        break;
    } /* end switch */
  } /* end if */
} /* end postEval */
```

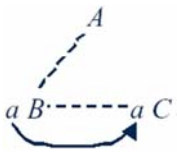
Inherited Attributes

- Definition
  - An attribute that is not synthesized is called an inherited attribute
- Different kinds of inherited dependencies
  - Inherited from parent to siblings



  - Inherited from sibling to sibling

  - Sibling inheritance via sibling pointers



Dept. of Computer Science at Sogang University

Inherited Attributes

- Inherited attributes can be computed by a preorder traversal, or combined preorder/inorder traversal of the parse or syntax tree
  - **procedure** *PreEval* (*T* : *treenode*)**;**

    begin

      for *each child C of T* **do**

        *compute all inherited attributes of C***;**

        *PreEval(C)***;**

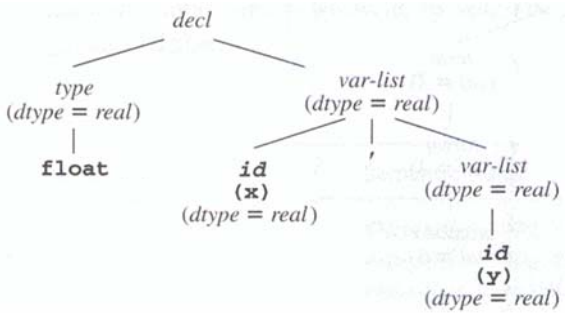
    end;
  - Unlike synthesized attributes, the order in which the inherited attributes of the children are computed is important
    - Inherited attributes may have dependencies among the attributes of the children

Grammar with inherited attribute ***dtype***

- Grammar
  - *decl* → *type var-list*
  - *type* → ***int*** | ***float***
  - *var-list* → ***id***, *var-list* | ***id***
- Attribute grammar

Grammar Rule	Semantic Rules
decl → *type var-list*	*var-list.dtype* = *type.dtype*
type → `int`	*type.dtype* = *integer*
type → `float`	*type.dtype* = *real*
$var\text{-}list_1$ → `id`, $var\text{-}list_2$	`id`.*dtype* = $var\text{-}list_1$.*dtype*
	$var\text{-}list_2$.*dtype* = $var\text{-}list_1$.*dtype*
var-list → `id`	`id`.*dtype* = *var-list.dtype*

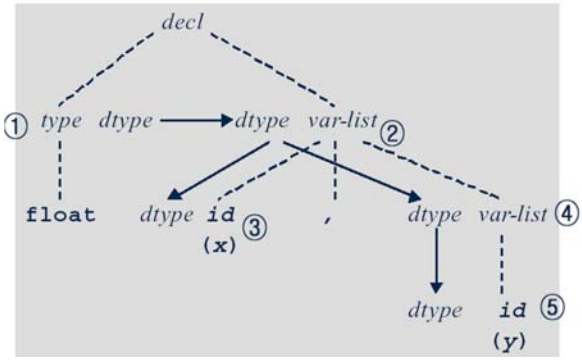
- Parse tree for the string ***float x,y***



Dept. of Computer Science at Sogang University

Grammar with inherited attribute ***dtype***

- Parse tree showing traversal order
  - Grammar
    - *decl* → *type var-list*
    - *type* → ***int*** | ***float***
    - *var-list* → ***id***, *var-list* | ***id***



- To compute the ***dtype*** attribute

```
procedure EvalType( T: treenode );
begin
  case nodekind of T of
  decl:
    EvalType( type child of T );
    Assign dtype of type child of T to var-list child of T;
    EvalType( var-list child of T );
  type:
    if child of T = int then T.dtype := integer
    else T.dtype := real;
  var-list:
    assign T.dtype to first child of T;
    if third child of T is not nil then
      assign T.dtype to third child of T;
      EvalType( third child of T );
  end case;
end EvalType;
```

Structure of a Syntax Tree

- C definition

```
typedef enum {decl,type,id} nodeKind;
typedef enum {integer,real} typeKind;
typedef struct treeNode
      { nodekind kind;
        struct treeNode *lchild, *rchild, *sibling;
        typekind dtype;
        /* for type and id nodes */
        char * name;
        /* for id nodes only */
      } *SyntaxTree;
```

- C code for *EvalType*
  - *Refer to the AST at page 281 in the textbook*

```
void evalType(SyntaxTree t)
{ switch (t->kind)
  { case decl:
      t->rchild->dtype = t->lchild->dtype;
      evalType(t->rchild);
      break;
    case id:
      if (t->sibling != NULL)
      { t->sibling->dtype = t->dtype;
        evalType(t->sibling);
      }
      break;
  } /* end switch */
} /* end evalType */
```

Dept. of Computer Science at Sogang University

Grammar with inherited attribute ***base*** and synthesized attribute **val**

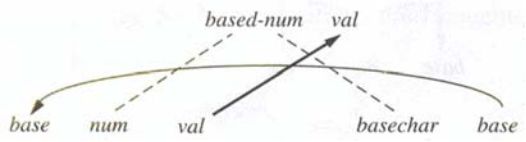
Grammar

- *based-num* → *num basechar*
- *basechar* → ***o*** | ***d***
- *num* → *num digit* | *digit*
- *digit* → ***0*** | ***1*** | ***2*** | ***3*** | ... | ***8*** | ***9***

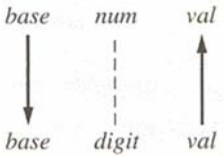
Grammar Rule	Semantic Rules
based-num → *num basechar*	*based-num.val* = *num.val* *num.base* = *basechar.base*
basechar → o	*basechar.base* = 8
basechar → d	*basechar.base* = 10
num_1 → num_2 *digit*	num_1 *.val* = **if** *digit.val* = *error* **or** num_2 *.val* = *error* **then** *error* **else** num_2 *.val* * num_1 *.base* + *digit.val* num_2 *.base* = num_1 *.base* *digit.base* = num_1 *.base*
num → *digit*	*num.val* = *digit.val* *digit.base* = *num.base*
digit → 0	*digit.val* = 0
digit → 1	*digit.val* = 1
...	...
digit → 7	*digit.val* = 7
digit → 8	*digit.val* = **if** *digit.base* = 8 **then** *error* **else** 8
digit → 9	*digit.val* = **if** *digit.base* = 8 **then** *error* **else** 9

Grammar with inherited attribute ***base*** and synthesized attribute ***val***

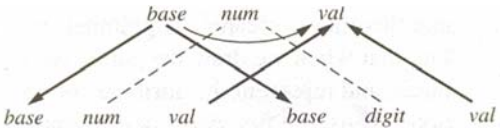
- *based-num → num basechar*



- *num → digit*

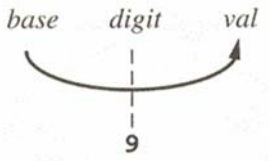


- *num → num digit*



$num_1.val =$
  if $digit.val = error$ **or** $num_2.val = error$
  then *error*
  else $num_2.val * num_1.base + digit.val$
$num_2.base = num_1.base$
$digit.base = num_1.base$

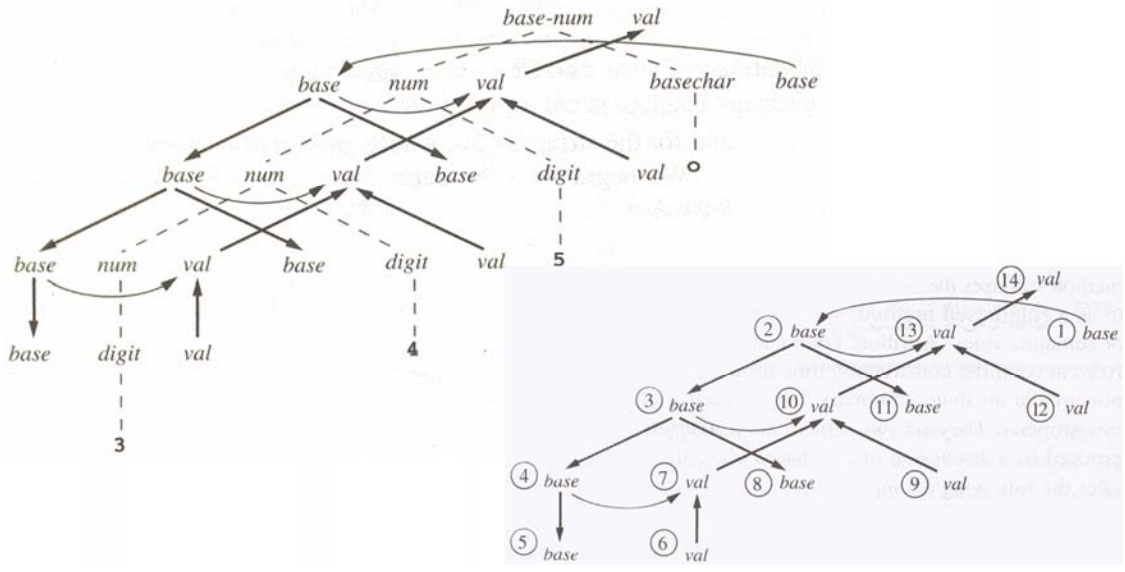
- *digit → **0** | **1** | **2** | **3** | ... | **8** | **9***



Dept. of Computer Science at Sogang University

Grammar with inherited attribute ***base*** and synthesized attribute ***val***

- Dependency graph for the string 345o
  - 12 6 9 1 2 11 3 8 4 5 7 10 13 14



Grammar with inherited attribute ***base*** and synthesized attribute ***val***

```
procedure EvalWithBase ( T: treenode );
begin
  case nodekind of T of
  based-num:
      EvalWithBase ( right child of T );
      assign base of right child of T to base of left child;
      EvalWithBase ( left child of T );
      assign val of left child of T to T.val;
  num:
      assign T.base to base of left child of T;
      EvalWithBase ( left child of T );
      if right child of T is not nil then
        assign T.base to base of right child of T;
        EvalWithBase ( right child of T );
        if vals of left and right children ≠ error then
            T.val := T.base*(val of left child) + val of right child
        else T.val := error;
      else T.val := val of left child;
  basechar:
      if child of T = o then T.base := 8
      else T.base := 10;
  digit:
      if T.base = 8 and child of T = 8 or 9 then T.val := error
      else T.val := numval ( child of T );
  end case;
end EvalWithBase;
```

Grammar

based-num → *num basechar*
basechar → ***o*** | ***d***
num → *num digit* | *digit*
digit → ***0*** | ***1*** | ***2*** | ***3*** | ... | ***8*** | ***9***

Dept. of Computer Science at Sogang University

Algorithm for Attribute Grammar

- In attribute grammar,
  - If the synthesized attributes depend on the inherited attributes but the inherited attributes do not depend on any synthesized attributes
  - Then possible to compute all the attributes in a single pass over parse or syntax tree
    - Combine the *PostEval* and *PreEval* pseudocode procedures

```
procedure CombinedEval ( T: treenode );
begin
  for each child C of T do
      compute all inherited attributes of C;
      CombinedEval ( C );
  compute all synthesized attributes of T;
end;
```