SEMANTIC ANALYSER

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Semantic Analyser

- Parser verifies that a program is syntactically correct and constructs a syntax tree (or other intermediate representation).
- **Semantic analyzer** checks that the program satisfies all other *static* language requirements that is if the structure verified by the parser makes any sense:

  1. **Is meaningful.**
  2. **Also it collects and computes information.**

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Semantic Analyser

- Checking for "correct meaning"
  - Warn about dubious meaning
- Long-distance and deep relations
  - Lexer and parser are only short-distance
- Implementation could be using AST traversal

Attributes

1. **Data type of a variable.**
2. **Value of an expression.**
3. **Location of a variable in memory.**
4. **Object code of a function/procedure.**
5. **Number of significant digits in a number.**

Attribute Grammar/Equation

$$X_i.a_j = f_{ij}\ (X_0.a_1, X_0.a_2, \ldots X_0.a_k, X_1.a_1 \ldots X_n.a_1 .. X_n.a_k)$$

Different instances of the same non-terminal must be subscripted to be distinguished.

Dr. D. M. Akbar Hussain
Department of Electronic Systems
5

Syntax Directed Semantics

Attribute grammar are most useful for languages that obey the **syntax directed semantics,** which fortunately most languages do, but the designers have to write the attribute grammars by hand there is no standard way or tool available for this.

Dr. D. M. Akbar Hussain
Department of Electronic Systems
6

Semantic Rules

Grammar:

exp → *exp* **+** *term* | *exp* **-** *term* | *term*
term → *term* ***** *factor* | *factor*
factor → **(** *exp* **)** | **number**

GRAMMAR RULE	SEMANTIC RULES
$exp_1 \rightarrow exp_2$ + $term$	$exp_1.val = exp_2.val + term.val$
$exp_1 \rightarrow exp_2$ - $term$	$exp_1.val = exp_2.val - term.val$
$exp \rightarrow term$	$exp.val = term.val$
$term_1 \rightarrow term_2$ * $factor$	$term_1.val = term_2.val * factor.val$
$term \rightarrow factor$	$term.val = factor.val$
$factor \rightarrow$ (exp)	$factor.val = exp.val$
$factor \rightarrow$ **number**	$factor.val =$ **number**$.val$

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Grammar Rules for decimal & octal numbers

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

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Attribute Grammar for decimal & octal numbers

Grammar Rules	Semantic Rules
based-num → num base-char	based-num.val = num.val num.base = base-char.base
base-char → o	base-char.base = o
base-char → d	base-char.base = d
num1 → num2 digit	num1.val = if digit.val = error or num2.val = error then error else num2.val * num1.base + digit.val num2.base = num1.base digit.base = num1.base
num → digit	num.val = digit.val digit.base = num.base
digit → 0	digit.val = 0
digit → 1	digit.val = 0
digit → 2	digit.val = 0
………………..	…………………………….
digit → 7	digit.val = 0
digit → 8	digit.val = if digit.base = 8 then error else 8
digit → 9	digit.val = if digit.base = 8 then error else 9

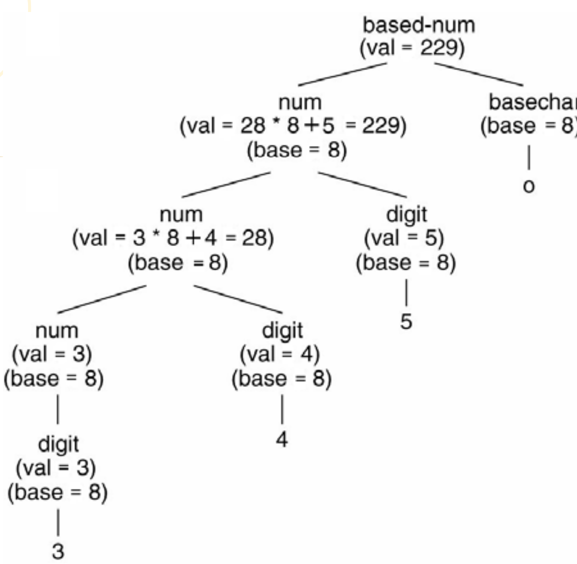
based-num → num basechar
basechar → *o* | *d*

num → num *digit* | *digit*

digit → *0* | *1* | *2* | *3* … | *9*

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Example: 345 o



Dr. D. M. Akbar Hussain
Department of Electronic Systems

Algorithms for Attribute Computation

$X_i .a_j = f_{ij} \; (X_0 .a_1 , X_0 .a_2 , ... X_0 .a_k ,X_1 .a_1 ..... X_n .a_1 ..X_n .a_k)$

- **Dependency Graphs**
  - **Dependency graph indicates order in which attributes must be computed.**
- **Evaluation Order**

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Constructing Dependency Graphs

1. **Parse Tree Method**
2. **Rule Based Method**
3. **Oblivious Method**

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Constructing Dependency Graphs

Parse tree method. At compile time, this method obtain an evaluation order from a *topological sort* of the dependency graph constructed from the parse tree for each input. This method will fail to find an evaluation order only if the dependency graph for the particular parse tree under construction has a cycle.

Rule based method. At compiler construction time, the semantic rules associated with productions are analyzed, either by hand, or by specialized tool. For each production, the order in which the attributes associated with that production are evaluated is predetermined at compiler construction time.

Oblivious method. An evaluation order is chosen without considering the semantic rules. For example, if translation takes place during parsing, then the order of evaluation is forced by the parsing method, independent of the semantic rules. An oblivious evaluation order restricts the class of syntax directed definitions that can be implemented.

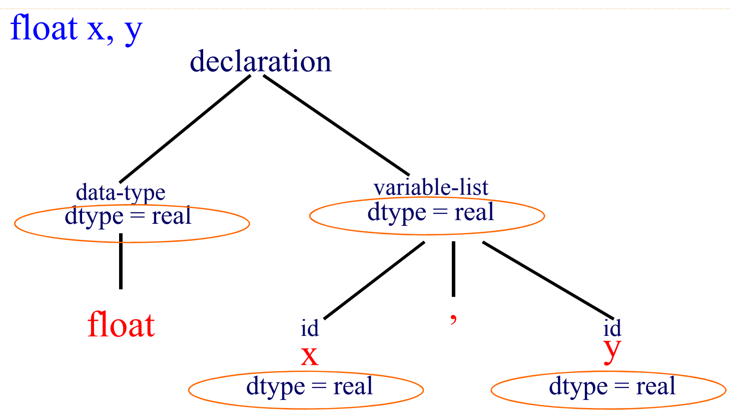
Attribute Grammar from Grammar Rules

declaration → data-type variable-list
data-type → *int* | *float*
variable-list → *id* , variable-list | *id*

Grammar Rules	Semantic Rules
declaration → data-type variable-list	variable-list.*dtype* = type.*dtype*
data-type → int	type.*dtype* = integer
data-type → float	type.*dtype* = real
variable-list1 → id , variable-list2	id.*dtype* = variable-list1.*dtype* variable-list2.*dtype* = variable-list1.*dtype*
variable-list → id	id.*dtype* = variable-list.*dtype*

Parse Tree

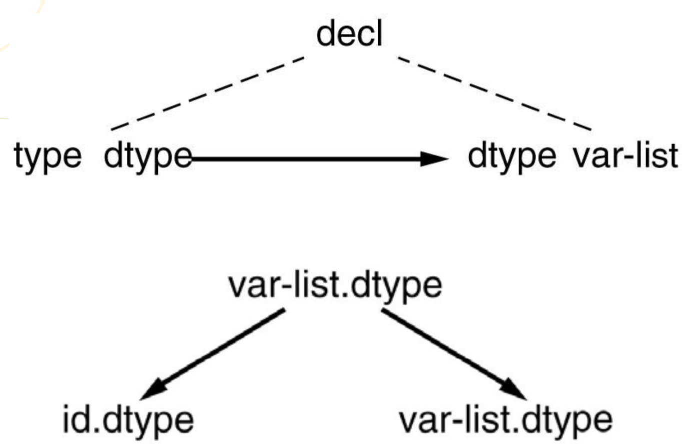
declaration → data-type variable-list
data-type → *int* | *float*
variable-list → *id* , variable-list | *id*



Dr. D. M. Akbar Hussain
Department of Electronic Systems

15

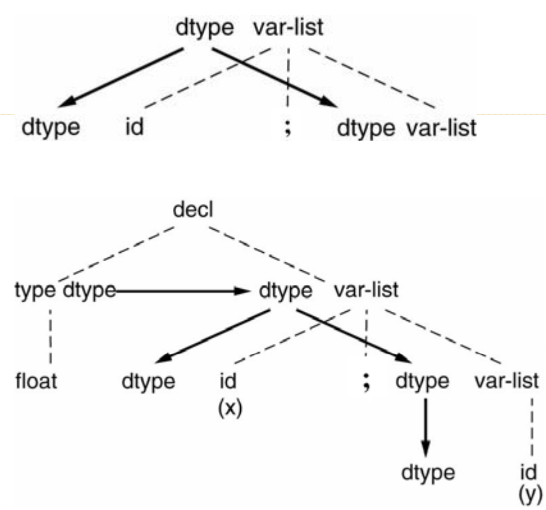
Dependency Graph



Dr. D. M. Akbar Hussain
Department of Electronic Systems

16

Dependency Graph

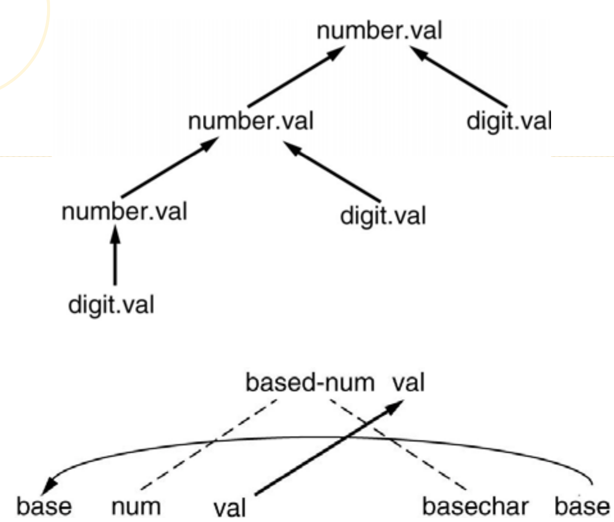


Dr. D. M. Akbar Hussain

Department of Electronic Systems

17

Dependency Graph

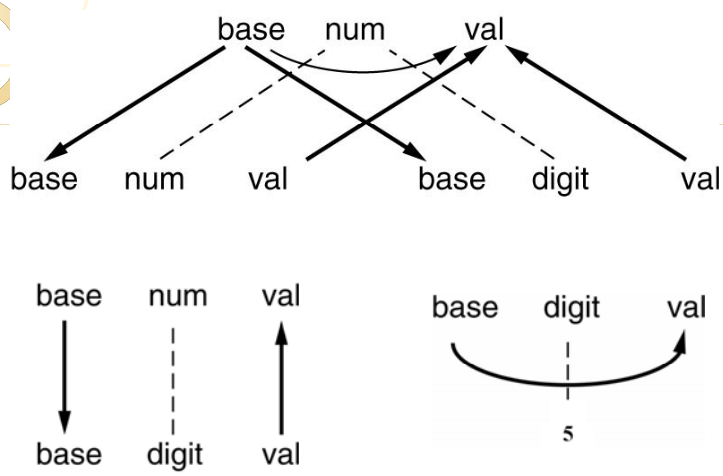


Dr. D. M. Akbar Hussain

Department of Electronic Systems

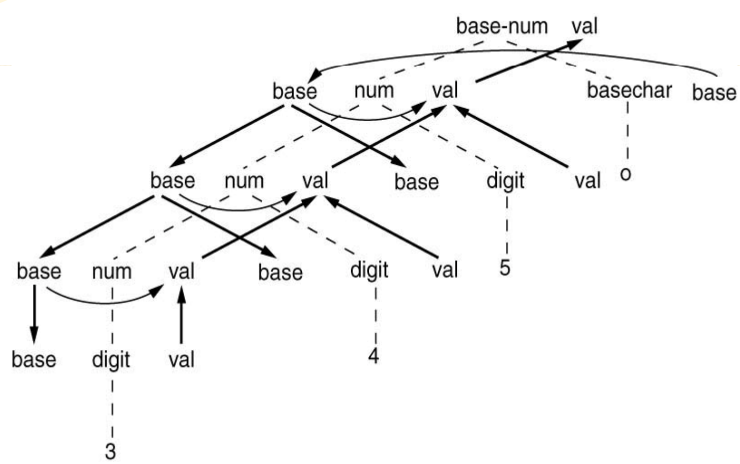
18

Dependency Graph



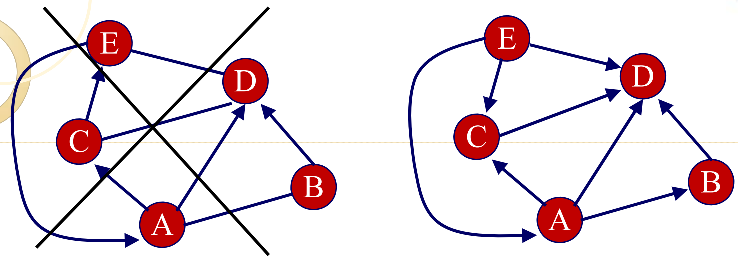
Dr. D. M. Akbar Hussain
Department of Electronic Systems

Dependency Graph



Dr. D. M. Akbar Hussain
Department of Electronic Systems

Diagraph



A digraph is a graph whose edges are all directed . Short for "directed graph".

Applications:

- One-way streets.
- Flights.
- Task scheduling.

Dr. D. M. Akbar Hussain
Department of Electronic Systems

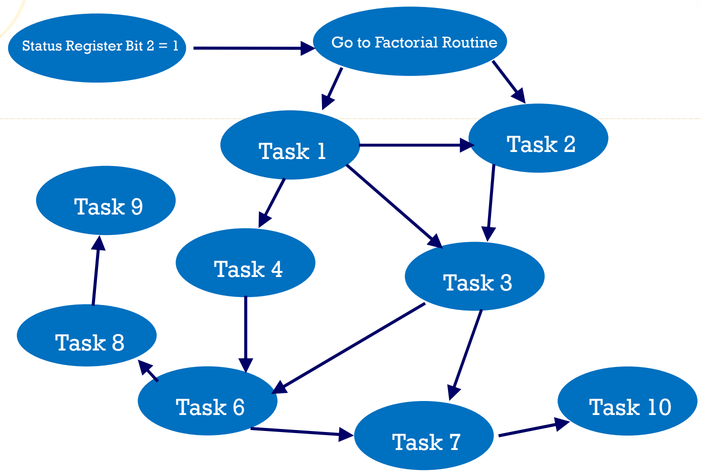
DAG: Directed Acyclic Graph

A directed acyclic graph (DAG) is a **digraph** that has no directed cycles.

A Topological Sort finds a path from u to v (if the path exist) in such a way that u always appear before v.

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Scheduling Example



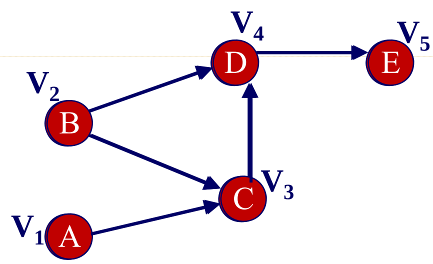
Dr. D. M. Akbar Hussain
Department of Electronic Systems
23

Topological Ordering

A topological ordering of a digraph is a numbering v1 , …, vn of the vertices such that for every edge (vi , vj), we have **i < j** .

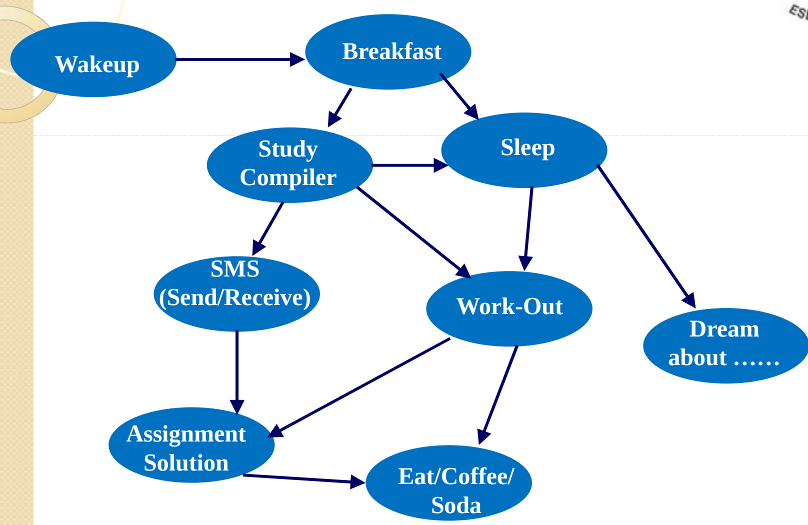
Dr. D. M. Akbar Hussain
Department of Electronic Systems
24

Diagraph & Topological Ordering



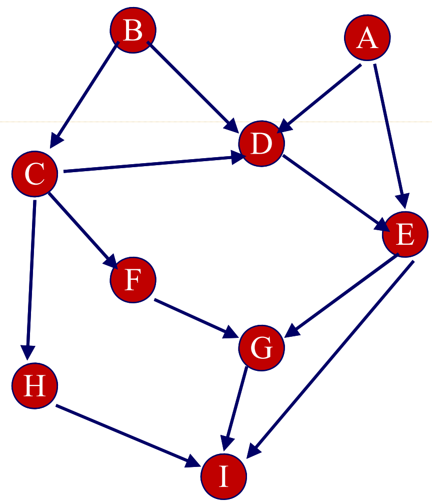
Dr. D. M. Akbar Hussain
Department of Electronic Systems
25

Topological Sorting

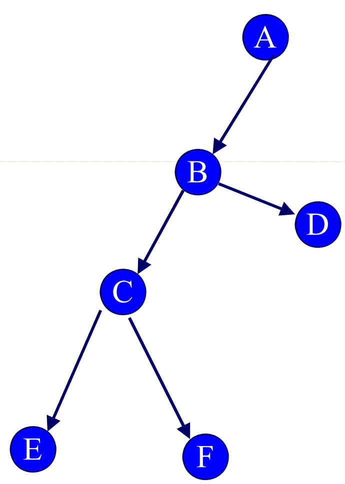


Dr. D. M. Akbar Hussain
Department of Electronic Systems
26

Example

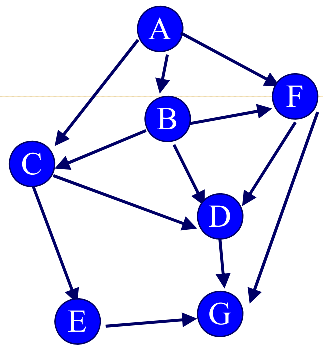


Topological Sort Example



A B D C E F, A B D C F E and A B C E F D …….. are valid topological sorted orders.

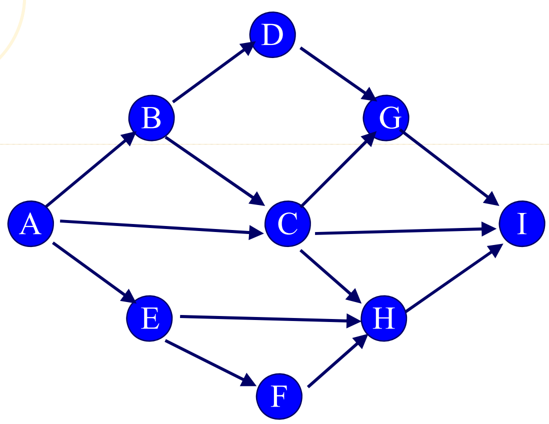
Topological Sort Example



Topological sort is not unique.
In this graph, A B C F D E G, A B F C E D G and A B C F E D G are valid topological sorted orders.

Dr. D. M. Akbar Hussain
Department of Electronic Systems
29

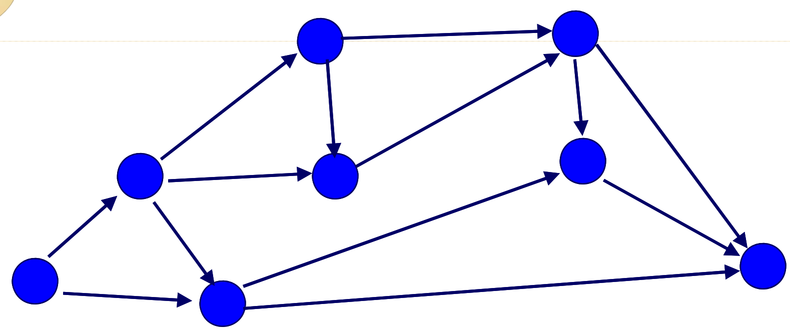
Topological Sort Exercise



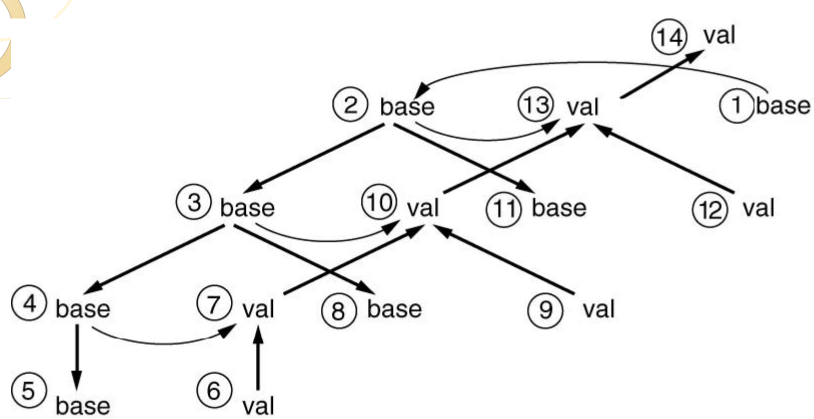
Determine Topological Order for this graph.

Dr. D. M. Akbar Hussain
Department of Electronic Systems
30

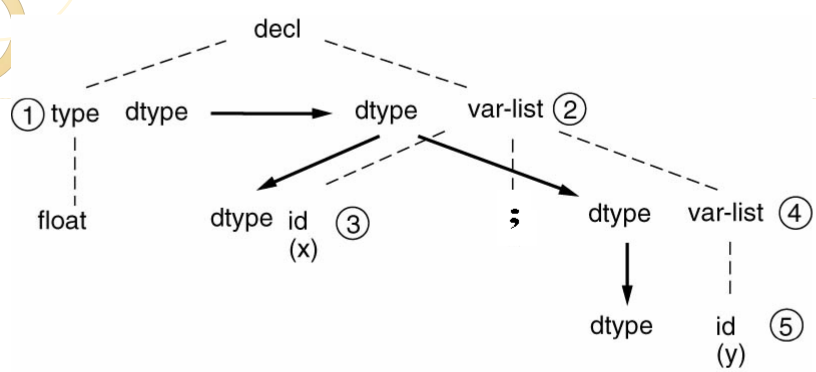
Topological Sort Exercise



Topological Sort



Topological Sort



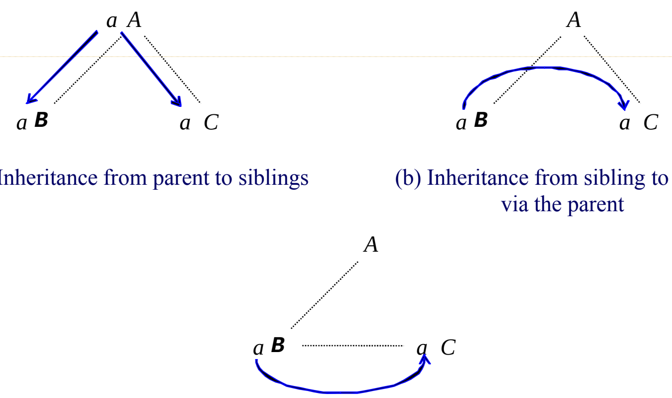
Dr. D. M. Akbar Hussain
Department of Electronic Systems
33

Type of Attributes

- **Synthesized attributes:**
  - Synthesized attributes always flow from children to parents, and can always be computed by a post-order traversal.
  - An attribute grammar in which all attributes are synthesized is called S-attributed definition.
  - An other class of SDD is called L-attributed definition, in which the dependency graph edges (evaluation order) goes from left to right only. Which some time necessary to use.
- **Inherited attributes:**
  - Inherited attributes can flow any other way.

Dr. D. M. Akbar Hussain
Department of Electronic Systems
34

Different Dependency Relationship



(a) Inheritance from parent to siblings

(b) Inheritance from sibling to sibling via the parent

(c) Sibling inheritance via sibling pointers

Dr. D. M. Akbar Hussain
Department of Electronic Systems

35

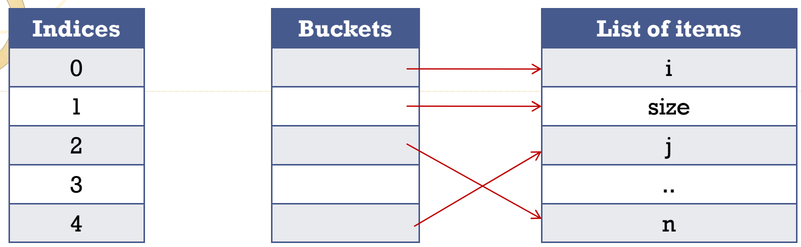
Symbol Table

- **Structure of Symbol Table**
  - Link List, Hash Table etc.
    - **Hash Function. (watch out for collisions)**
      1. **Open Addressing. (Inserting new items in successive bucket)**
      2. **Separate Chaining.(Each bucket is a list so when collision occurs new item is added to the list)**
- ✓ Declaration.
- ✓ Scope Rules and Block Structure.
- ✓ Example of Symbol Table.

Dr. D. M. Akbar Hussain
Department of Electronic Systems

36

Hash Table



Hash table is an array of entries, called buckets indexed by an integer range, A hash function compute the search key (identifier name) into an integer hash value (indices), and the corresponding info is stored in the bucket.

Scope of an Identifier

1. The part of the program in which the identifier is accessible or visible.
2. An identifier may have restricted scope.
3. Same identifier may refer to different things in different parts of the program.
4. Different scopes for same name don't overlap.
5. Not all kinds of identifiers follow the most-closely nested rule

Scope

```
class Foo {
  int value = 39;
  test() {
    int b = 12;
    b = value + b;
  }
  setValue(int c) {
    value = c;
    int d = c; {
       c = c + d;
       value = c;
    }
  }
}
public class Bar {
  int newvalue = 42;
  setValue(int c){
       newvalue = c;
  }
}
```

scope of b

scope of value

scope of c

scope of d

scope of c

scope of newvalue

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Symbol Table

```
class foo {
  int a = 39;
  test();
  int b = a + 3;
}
```

Symbol	Kind	Type
foo	class	int
a	var	int
test	method	int
b	var	int

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Symbol Table Lookup

Symbol	Kind	Type
foo	method	int
a	var	int
test	method	int
b	var	int

(test)

Symbol	Kind	Type
test	method	int

Symbol	Kind	Type
a	var	int

```
int foo () {
  int a = 39;
  test();
  int b = a + 3;
}
```

Lookup (b)

Symbol	Kind	Type
b	var	int

Symbol Table Lookup

Symbol	Kind	Type
setValue	method	int
c	var	int
test	method	int
value	var	int
d	var	int

(test)

Symbol	Kind	Type
b	var	int

(setValue)

Symbol	Kind	Type
d	var	int

(setValue-block)

Symbol	Kind	Type
c	var	int

Lookup (value)

```
setValue(int c)
  {
    test ();
    int value = c;
    int d = c;
    {
      c = c + d;
      value = c;
    }
  }
```

Symbol Table Lookup

ERROR !

Symbol	Kind	Type
setValue	method	int
c	var	int
test	method	int
value	var	int
d	var	int

(test)

Symbol	Kind	Type
b	var	int

(setValue)

Symbol	Kind	Type
d	var	int

(setValue-block)

Lookup (myValue)

Symbol	Kind	Type
c	var	int

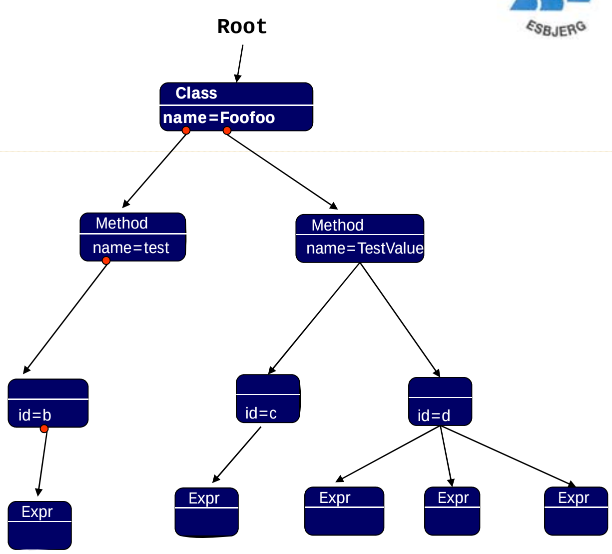
```
setValue(int c)
  {
    test ();
    int value = c;
    int d = c;
    {
      c = c + d;
      myValue = c;
    }
  }
```

Dr. D. M. Akbar Hussain
Department of Electronic Systems

43

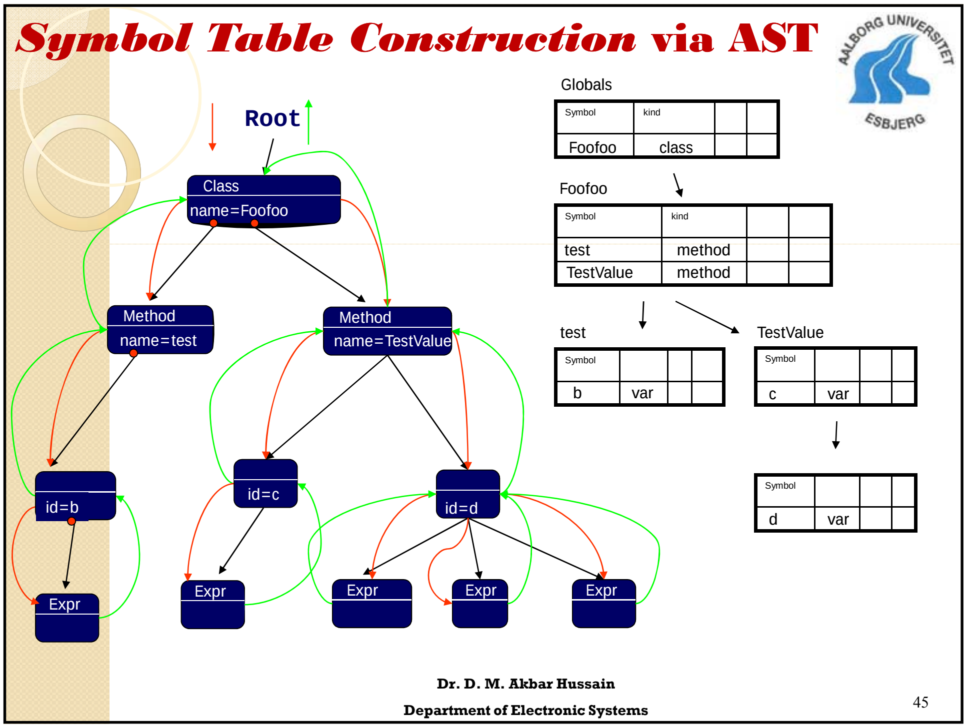
Symbol Table Construction

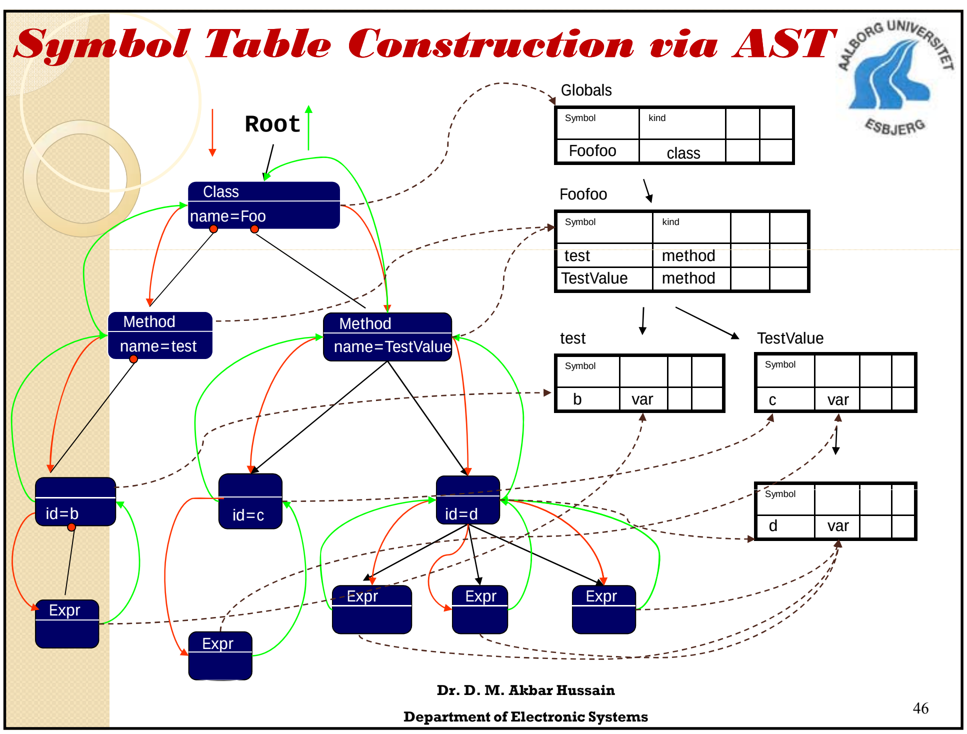
```
class Foofoo {

  int test() {
    int b = 3;
    b = b + 3;
  }
  int TestValue(){
    int c = 10 + 4;
    int d = 8; {
       c = c + d;
       d = d + c;
       c = d + 10;
    }
  }
}
```



Dr. D. M. Akbar Hussain
Department of Electronic Systems

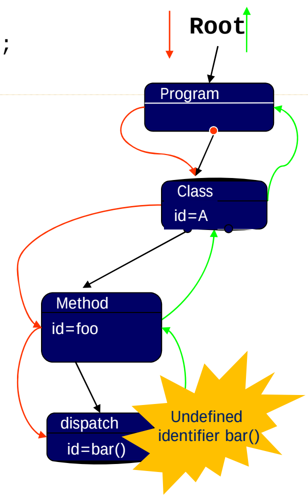
44


Symbol Table Construction via AST
Root
Class
name=Foofoo
Method
name=test
Method
name=TestValue
id=b
id=c
id=d
Expr
Globals
Symbol kind
Foofoo class
Foofoo
test method
TestValue method
test
b var
TestValue
c var
d var
Dr. D. M. Akbar Hussain
Department of Electronic Systems
45


Symbol Table Construction via AST
Root
Class
name=Foo
Method
name=test
Method
name=TestValue
id=b
id=c
id=d
Expr
Globals
Symbol kind
Foofoo class
Foofoo
test method
TestValue method
test
b var
TestValue
c var
d var
Dr. D. M. Akbar Hussain
Department of Electronic Systems
46

Symbol Table Construction via AST

```
class A {
   foo() {
         bar();
   }
}
```



globals

Symbol	kind
A	class

A

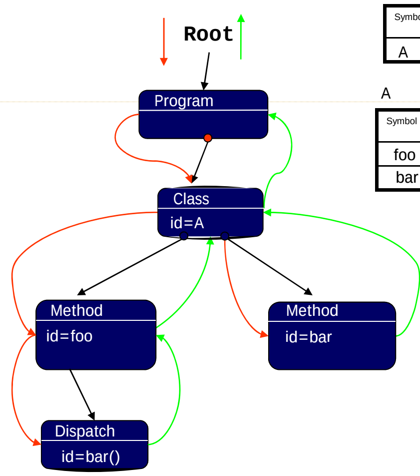
Symbol	kind
foo	method

Dr. D. M. Akbar Hussain
Department of Electronic Systems

47

Symbol Table Construction

```
class A {
   foo() {
         bar();
   }
   bar() {
         …
   }
}
```



globals

Symbol	kind
A	class

A

Symbol	kind	FREF
foo	method	
bar	method	true

Dr. D. M. Akbar Hussain
Department of Electronic Systems

48

TINY Symbol Table

```
1:    { Simple Program
2:    in Tiny Language –
3:    computing factorial
4:    }
5:    read x; { Input an Integer }
6:    if 0 < x then { don't compute if x <= 0 }
7:            fact := 1;
8:            repeat
9:                    fact := fact * x;
10:                   x := x – 1;
11:           until x = 0;
12:           write fact { output factorial of x }
13:   end
```

Dr. D. M. Akbar Hussain
Department of Electronic Systems

TINY Symbol Table

Variable Name	Location	Line Numbers
x	0	5 6 9 10 10 11
fact	1	7 9 9 12

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Yacc Example (Synthesized Attribute)

	Parsing Stack	Input	Parsing Action	Value Stack	Semantic Action
1	$	3 * 4 + 5 $	Shift	$	
2	$ n	* 4 + 5 $	Reduce	$ n	E.val = n.val
3	$ E	* 4 + 5 $	Shift	$ 3	
4	$ E *	4 + 5 $	Shift	$ 3 *	
5	$ E * n	+ 5 $	Reduce	$ 3 * n	E.val = n.val
6	$ E * E	+ 5 $	Reduce	$ 3 * 4	E_1.val = E_2.val * E_3.val
7	$ E	+ 5 $	Shift	$ 12	
8	$ E +	5 $	Shift	$ 12 +	
9	$ E + n	$	Reduce	$ 12 + n	E.val = n.val
10	$ E + E	$	Reduce	$ 12 + 5	E_1.val = E_2.val * E_3.val
11	$ E	$		$ 17	

Attribute Computation

- **Attributes as Parameter/Return Values**

  For recursive procedures/functions Inherited attributes can be passed as argument parameters and Synthesized attributes as return values

Data Types & Type Checking

- Type Expressions and Type Constructors
  - Array
  - Record
  - Union
  - Pointer
  - Function
  - Class
- Type Names, Type Declaration and Recursive Types
- Type Equivalence
- Type Inference and Type Checking

Dr. D. M. Akbar Hussain
Department of Electronic Systems

Additional Topics in Type Checking

- Overloading
  - *Same operator name is used for two different operations.*
- Type Conversion and Coercion
  - *A common type must be found for mixed types.*
- Polymorphic Typing
  - *Language constructs have more than one type.*

Dr. D. M. Akbar Hussain
Department of Electronic Systems