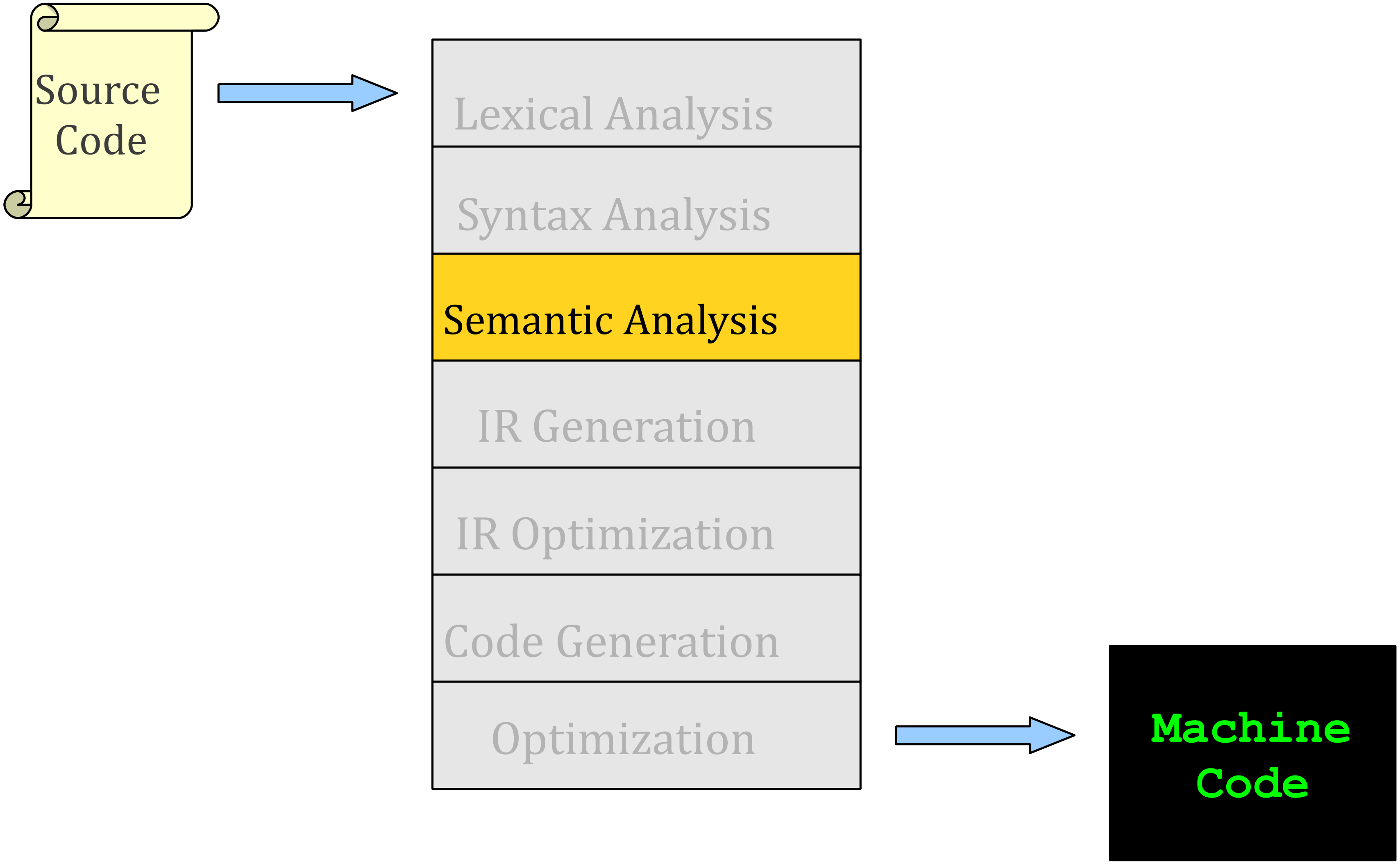
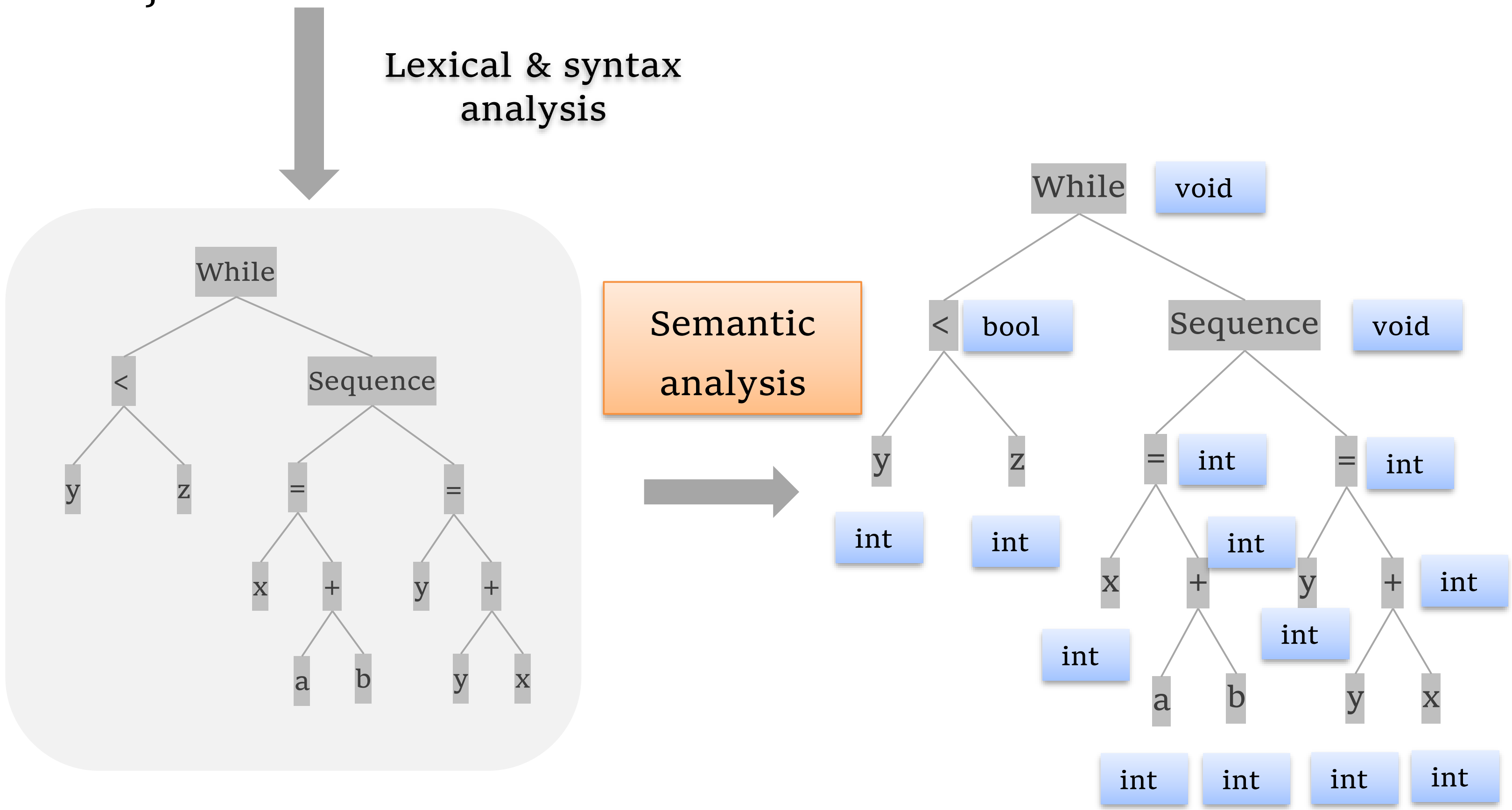
Lecture 06
Semantic Analysis

Where We Are


Source Code
Lexical Analysis
Syntax Analysis
Semantic Analysis
IR Generation
IR Optimization
Code Generation
Optimization
Machine Code

```
while (y < z) {
    int x = a + b;
    y += x;
}
```



Where We Are

- Program is lexically well-formed:
- Program is syntactically well-formed:
  - Have the correct structure/ syntactically valid.
- Does this mean that the program is legal?

A Short Program

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x = new string;

        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

A Short Program

```
class MyClass implements MyInterface {
    string myInteger;

    void doSomething() {
        int[] x = new string;
        x[5] = myInteger * y;
    }
    void doSomething() {

    }
    int fibonacci(int n) {
        return doSomething() + fibonacci(n - 1);
    }
}
```

Interface not declared

Can't multiply strings

Wrong type

Variable not declared

Can't redefine functions

Can't add void

No main function

Semantic Analysis

- Ensure that the program has a well-defined **meaning**.
- Verify properties of the program that aren't caught during the earlier phases:
  - Variables are declared before they're used.
  - Expressions have the right types.
  - Classes don't inherit from nonexistent base classes
  - ...
- Once we finish semantic analysis, we know that the user's input program is legal.

Typical examples of Semantic Analysis

a) Type Checking:

- Whether the types of operands of a operator are equal?
- Whether the types of the left and right hand side of assignment are equal?
- Whether the type of index of array is proper?

b) Others:

- Whether an identifier used has been declared?
- Has V been declared to be a variable of array type for "V[E]" ?

Limitations of CFGs

- Using **CFGs**:
  - How would you prevent duplicate class definitions?
  - How would you differentiate variables of one type from variables of another type?
- For most programming languages, these are *provably impossible*.
- **Attribute Grammars** are used to describe the semantic rules for semantic analysis.

Outline

- Semantic Analysis
  - Attributes and Attribute Grammars
  - Dependency Graphs and Algorithms for Attribute Computation
  - Symbol Table and Scope Checking
  - Type Checking for Semantic Analysis of a Program

I. Attributes and Attribute Grammars

- Attribute grammars are used to describe the semantic rules
- Definition of Attribute: An attribute is any property of a programming language construct
- Typical examples of attributes are:
  - **The data type of a variable**
  - **The value of an expression**
  - **The object code of a procedure**

Attribute

- Attributes are associated directly with the grammar symbols (terminals and nonterminals)

- If X is a grammar symbol, and *a* is an attribute associated to X, then the value of *a* associated to X is written as X.*a*

Attribute Equation (or Semantic Rule)

- Given a collection of attributes $a_1, \ldots, a_k$,

for each grammar rule $X_0 \rightarrow X_1\ X_2 \ldots X_n$, the values of the attributes $X_i.a_j$ of each grammar symbol X_i are related to the values of the attributes of the other symbols in the rule

- Attribute equation has the form

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

where f_{ij} is a mathematical function of its arguments

grammar rule	Attribute equation
number1->number2 digit	**number1.val=number2.val*10+digit.val**

Attribute Grammar

- An attribute grammar for the attributes a1,…,ak is the collection of all attribute equations, for all the grammar rules of the language
- Typically, attribute grammars are written in tabular form

Grammar Rule	Semantic Rules
Rule 1	Associated attribute equations
…	
Rule n	Associated attribute equations

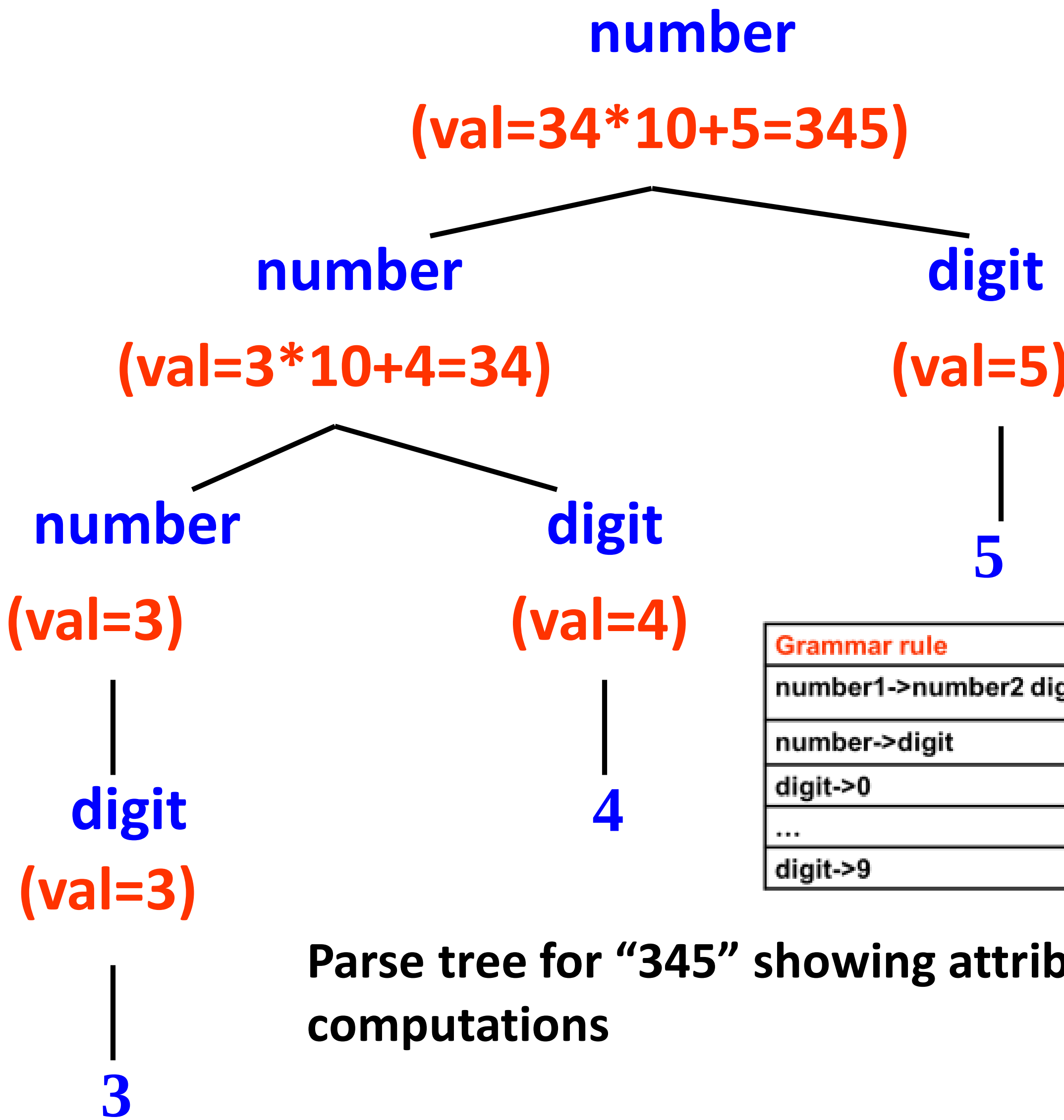
Example 1

Attribute grammar for unsigned numbers

Attribute of a number is its value

Grammar rule	Semantic Rule
number1->number2 digit	number1.val=number2.val*10+digit.val
number->digit	number.val=digit.val
digit->0	digit.val=0
…	…
digit->9	digit.val=9

The meaning of the attribute equations for a particular string can be visualized using the parse tree for the string



Grammar rule	Semantic Rule
number1->number2 digit	number1.val=number2.val*10+digit.val
number->digit	number.val=digit.val
digit->0	digit.val=0
...	...
digit->9	digit.val=9

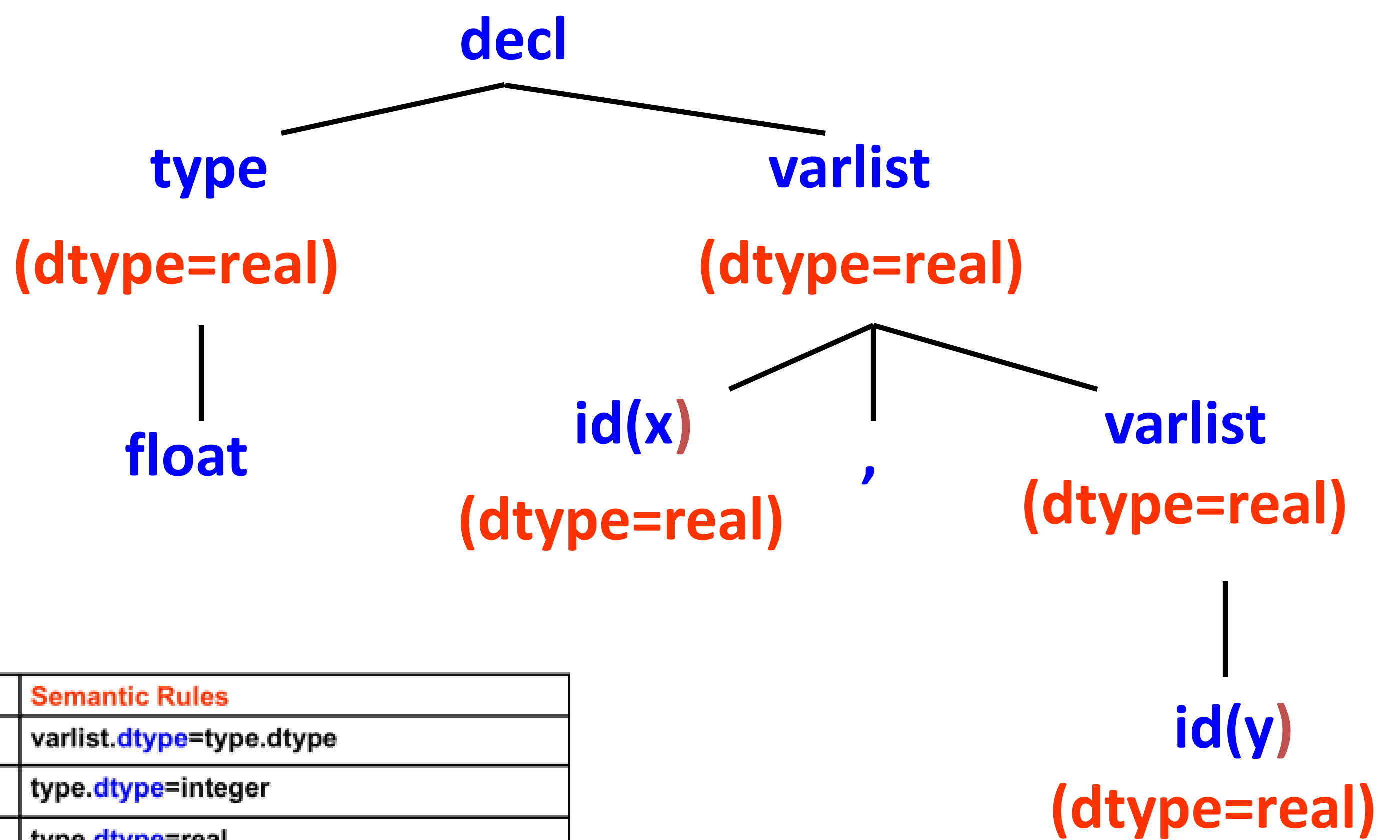
Parse tree for “345” showing attribute computations

Example 2

Attribute grammar for variable declarations
Attribute of the variable is data type

Grammar rule	Semantic Rules
decl->type varlist	varlist.dtype=type.dtype
type->int	type.dtype=integer
type->float	type.dtype=real
varlist1->id,varlist2	id.dtype=varlist1.dtype varlist2.dtype=varlist1.dtype
var-list->id	id.dtype=varlist.dtype

Parse tree for the string "float x,y" showing the dtype attribute



Grammar rule	Semantic Rules
decl->type varlist	varlist.dtype=type.dtype
type->int	type.dtype=integer
type->float	type.dtype=real
varlist1->id,varlist2	id.dtype=varlist1.dtype varlist2.dtype=varlist1.dtype
var-list->id	id.dtype=varlist.dtype