

# 6.035

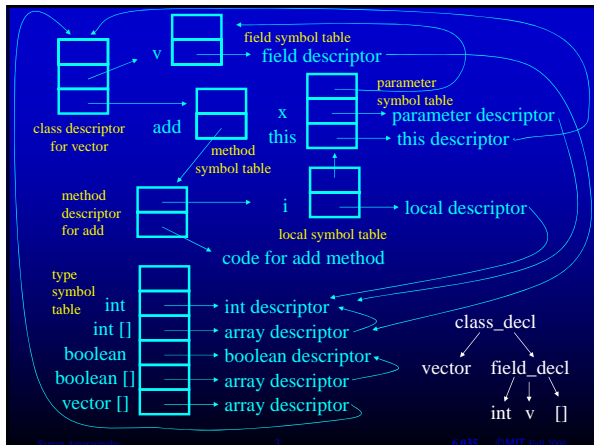
Fall 2006

## Semantic Analysis

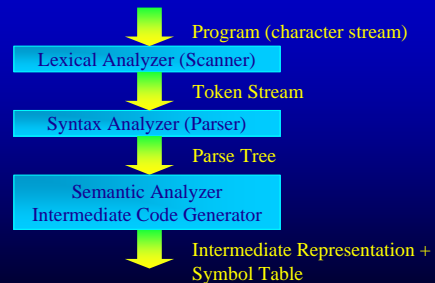
Saman Amarasinghe  
Massachusetts Institute of Technology

## Symbol Table Summary

- Program Symbol Table (Class Descriptors)
- Class Descriptors
  - Field Symbol Table (Field Descriptors)
    - Pointer to Field Symbol Table for SuperClass
  - Method Symbol Table (Method Descriptors)
    - Pointer to Method Symbol Table for Superclass
- Method Descriptors
  - Local Variable Symbol Table (Local Variable Descriptors)
    - Parameter Symbol Table (Parameter Descriptors)
      - Pointer to Field Symbol Table of Receiver Class
- Local, Parameter and Field Descriptors
  - Type Descriptors in Type Symbol Table or Class Descriptors



## Where are we?



## What is the semantics of a program?

- Syntax
  - How a program looks like
  - Textual representation or structure
  - A precise mathematical definition is possible
- Semantics
  - What is the meaning of a program
  - Harder to give a mathematical definition

## Why do semantic checking?

- Make sure the program conforms to the programming language definition
- Provide meaningful error messages to the user
- Don't need to do additional work, will discover in the process of intermediate representation generation

## Semantic Checking

- Static checks vs. Dynamic checks
- Static checks
  - Flow-of-control checks
  - Uniqueness checks
  - Type checks

Simon Arnsperg

7

6.035 ©MIT Fall 2005

## Flow of control checks

11

- Flow-control of the program is context sensitive
- Examples:
  - Declaration of a variable should be visible at use (in scope)
  - Declaration of a variable should be before use
  - Each exit path returns a value of the correct type
- What else?

Simon Arnsperg

8

6.035 ©MIT Fall 2005

## Uniqueness checks

- Use and misuse of identifiers
  - Cannot represent in a CFG (same token)
- Examples:
  - No identifier can be used for two different definitions in the same scope

Simon Arnsperg

9

6.035 ©MIT Fall 2005

## Type checks

15

- Most extensive semantic checks
- Examples:
  - Number of arguments matches the number of formals and the corresponding types are equivalent
  - If called as an expression, should return a type
  - Each access of a variable should match the declaration (arrays, structures etc.)
  - Identifiers in an expression should be "evaluatable"
  - LHS of an assignment should be "assignable"
  - In an expression all the types of variables, method return types and operators should be "compatible"

Simon Arnsperg

10

6.035 ©MIT Fall 2005

## Dynamic checks

17

- Array bounds check
- Null pointer dereference check

Simon Arnsperg

11

6.035 ©MIT Fall 2005

## Type Systems

17

- A type system is used to for the type checking
- A type system incorporates
  - syntactic constructs of the language
  - notion of types
  - rules for assigning types to language constructs

Simon Arnsperg

12

6.035 ©MIT Fall 2005

## Type expressions

- A compound type is denoted by a type expression
- A type expression is
  - a basic type
  - application of a type constructor to other type expressions

Simon Arnsperth

13

6.035 ©MIT Fall 2005

## Type Expressions: Basic types

18

- Atomic types defined by the language
- Examples:
  - integers
  - booleans
  - floats
  - characters
- `type_error`
  - special type that'll signal an error
- `void`
  - basic type denoting "the absence of a value"

Simon Arnsperth

14

6.035 ©MIT Fall 2005

## Type Expressions: Names

- Since type expressions may be named, a type name is a type expression

Simon Arnsperth

15

6.035 ©MIT Fall 2005

## Type Expressions: Products

- If  $T_1$  and  $T_2$  are type expressions  $T_1 \times T_2$  is also a type expression

Simon Arnsperth

16

6.035 ©MIT Fall 2005

## Type Expressions: Arrays

19

- If  $T$  is a type expressions a `array(T, I)` is also a type expression
  - $I$  is a integer constant denoting the number of elements of type  $T$
  - Example:

```
int foo[128];
array(integer, 128)
```

Simon Arnsperth

17

6.035 ©MIT Fall 2005

## Type Expressions: Function Calls

- Mathematically a function maps
  - elements of one set (the domain)
  - to elements of another set (the range)
- Example

```
int foobar(int a, boolean b, int c)
integer × boolean × integer × integer → integer
```

Simon Arnsperth

18

6.035 ©MIT Fall 2005

## Type Expressions: Some others

21

- Records
  - structures and classes
  - Example

```
class { int i; int j; }
integer × integer
```
- Functional Languages
  - functions that take functions and return functions
  - Example

```
(integer → integer) × integer → (integer → integer)
```

Simon St Laurent 19 6.035 ©MIT Fall 2005

## A simple typed language

- A language that has a sequence of declarations followed by a single expression

```
P → D; E
D → D; D | id : T
T → char | integer | array [ num ] of T
E → literal | num | id | E + E | E [ E ]
```

- Example Program

```
var: integer;
var + 1023
```

Simon St Laurent 20 6.035 ©MIT Fall 2005

## A simple typed language

- A language that has a sequence of declarations followed by a single expression

```
P → D; E
D → D; D | id : T
T → char | integer | array [ num ] of T
E → literal | num | id | E + E | E [ E ]
```

- What are the semantic rules of this language?

Simon St Laurent 21 6.035 ©MIT Fall 2005

## Parser actions

23

```
P → D; E
D → D; D
D → id : T      { addtype(id.entry, T.type); }
T → char       { T.type = char; }
T → integer    { T.type = integer; }
T → array [ num ] of T1
                { T.type = array(T1.type, num.val); }
```

Simon St Laurent 22 6.035 ©MIT Fall 2005

## Parser actions

```
E → literal    { E.type = char; }
E → num        { E.type = integer; }
E → id         { E.type = lookup_type(id.name); }
```

Simon St Laurent 23 6.035 ©MIT Fall 2005

## Parser actions

24

```
E → E1 + E2  { if E1.type == integer and
                  E2.type == integer then
                  E.type = integer
                  else
                  E.type = type_error
                  }
```

Simon St Laurent 24 6.035 ©MIT Fall 2005

## Parser actions

```
E → E1 [E2]  { if E2.type == integer and
                    E1.type == array(s, t) then
                      E.type = s
                    else
                      E.type = type_error
                    }
```

## Type Equivalence

25

- How do we know if two types are equal?
  - Same type entry

– Example:

```
int A[128];
foo(A);
```

```
foo(int B[128]) { ... }
```

- Two different type entries in two different symbol tables
- But they should be the same

## Structural Equivalence

- If the type expression of two types have the same construction, then they are equivalent
- “Same construction”
  - Equivalent base types
  - Same set of type constructors are applied in the same order (i.e. equivalent type tree)

## Type Coercion

26

- Implicit conversion of one type to another type
- Example

```
int A;
float B;
B = B + A
```
- Two types of coercion
  - widening conversions
  - narrowing conversions

## Narrowing conversions

- Conversions that may lose information
- Examples:
  - integers to chars
  - longs to shorts
- Rare in languages

## Widening conversions

- Conversions without loss of information
- Examples:
  - integers to floats
  - shorts to longs
- What is done in many languages (including decaf)

## Widening Conversions

- Basic Principle: Hierarchy of number types
  - int → float → double
- All coercions go up hierarchy
  - int to float;
  - int, float to double
- Result is type of operand highest up in hierarchy
  - int + float is float
  - int + double is double
  - float + double is double

Simon Arnsperth

31

6.035 ©MIT Fall 2005

## Type casting

- Explicit conversion from one type to another
- Both widening and narrowing
- Example

```
int A;
float B;
A = A + (int)B
```
- Unlimited typecasting can be dangerous

Simon Arnsperth

32

6.035 ©MIT Fall 2005

## Overloading

28

- Some operators may have more than one type.
- Example

```
int A, B, C;
float X, Y, Z;
A = A + B
X = X + Y
```
- Complicates the type system
  - Example

```
A = A + X
```

    - What is the type of + ?

Simon Arnsperth

33

6.035 ©MIT Fall 2005

## Parameter Descriptors

- When build parameter descriptor, have
  - name of type
  - name of parameter
- What is the check?
  - Is name of type identifies a valid type?
    - look up name in type symbol table
    - if not there, look up name in program symbol table (might be a class type)
    - if not there, fails semantic check

Simon Arnsperth

34

6.035 ©MIT Fall 2005

## Local Descriptors

- When build local descriptor, have
  - name of type
  - name of local
- What is the check?
  - Is name of type identifies a valid type?
    - look up name in type symbol table
    - if not there, look up name in program symbol table (might be a class type)
    - if not there, fails semantic check

Simon Arnsperth

35

6.035 ©MIT Fall 2005

## Local Symbol Table

- When build local symbol table, have a list of local descriptors
- What to check for?
  - duplicate variable names
  - shadowed variable names
- When to check?
  - when insert descriptor into local symbol table
- Parameter and field symbol tables similar

Simon Arnsperth

36

6.035 ©MIT Fall 2005

## Class Descriptor

- When build class descriptor, have
  - class name and name of superclass
  - field symbol table
  - method symbol table
- What to check?
  - Superclass name corresponds to actual class
  - No name clashes between field names of subclass and superclasses
  - Overridden methods match parameters and return type declarations of superclass

Simon Arnsperg

37

6.035 ©MIT Fall 2005

## Load Instruction

- What does compiler have? Variable name.
- What does it do? Look up variable name.
  - If in local symbol table, reference local descriptor
  - If in parameter symbol table, reference parameter descriptor
  - If in field symbol table, reference field descriptor
  - If not found, semantic error

Simon Arnsperg

38

6.035 ©MIT Fall 2005

## Load Array Instruction

- What does compiler have?
  - Variable name
  - Array index expression
- What does compiler do?
  - Look up variable name (if not there, semantic error)
  - Check type of expression (if not integer, semantic error)

Simon Arnsperg

39

6.035 ©MIT Fall 2005

## Add Operations

- What does compiler have?
  - two expressions
- What can go wrong?
  - expressions have wrong type
  - must both be integers (for example)
- So compiler checks type of expressions
  - load instructions record type of accessed variable
  - operations record type of produced expression
  - so just check types, if wrong, semantic error

Simon Arnsperg

40

6.035 ©MIT Fall 2005

## Type Inference for Add Operations

- Most languages let you add floats, ints, doubles
- What are issues?
  - Types of result of add operation
  - Coercions on operands of add operation
- Standard rules usually apply
  - If add an int and a float, coerce the int to a float, do the add with the floats, and the result is a float.
  - If add a float and a double, coerce the float to a double, do the add with the doubles, result is double

Simon Arnsperg

41

6.035 ©MIT Fall 2005

## Store Instruction

- What does compiler have?
  - Variable name
  - Expression
- What does it do?
  - Look up variable name.
    - If in local symbol table, reference local descriptor
    - If in parameter symbol table, error
    - If in field symbol table, reference field descriptor
    - If not found, semantic error
  - Check type of variable name against type of expression
    - If variable type not compatible with expression type, error

Simon Arnsperg

42

6.035 ©MIT Fall 2005

## Store Array Instruction

- What does compiler have?
  - Variable name, array index expression
  - Expression
- What does it do?
  - Look up variable name.
    - If in local symbol table, reference local descriptor
    - If in parameter symbol table, error
    - If in field symbol table, reference field descriptor
    - If not found, semantic error
  - Check that type of array index expression is integer
  - Check type of variable name against type of expression
    - If variable element type not compatible with expression type, error

Simon Arnsperth

43

6.035 ©MIT Fall 2005

## Method Invocations

- What does compiler have?
  - method name, receiver expression, actual parameters
- Checks:
  - receiver expression is class type
  - method name is defined in receiver's class type
  - types of actual parameters match types of formal parameters
  - What does match mean?
    - same type?
    - compatible type?

Simon Arnsperth

44

6.035 ©MIT Fall 2005

## Return Instructions

- What does compiler have?
  - Expression
- Checks:
  - If the return type matches the expression?

Simon Arnsperth

45

6.035 ©MIT Fall 2005

## Conditional Instructions

- What does compiler have?
  - Expression for the if-condition and the statement list of then (and else) blocks
- Checks:
  - If the conditional expression producing a Boolean value?

Simon Arnsperth

46

6.035 ©MIT Fall 2005

## Semantic Check Summary

- Do semantic checks when build IR
- Many correspond to making sure entities are there to build correct IR
- Others correspond to simple sanity checks
- Each language has a list that must be checked
- Can flag many potential errors at compile time

Simon Arnsperth

47

6.035 ©MIT Fall 2005