MIT 6.035
Semantic Analysis

Martin Rinard
Laboratory for Computer Science
Massachusetts Institute of Technology
Error Issue

• Have assumed no problems in building IR
• But are many static checks that need to be done as part of translation
• Called Semantic Analysis
Goal of Semantic Analysis

- Ensure that program obeys certain kinds of sanity checks
  - all used variables are defined
  - types are used correctly
  - method calls have correct number and types of parameters and return value
- Checked when build IR
- Driven by symbol tables
Symbol Table Summary

• Program Symbol Table (Class Descriptors)
• Class Descriptors
  – Field Symbol Table (Field Descriptors)
    • Field Symbol Table for SuperClass
  – Method Symbol Table (Method Descriptors)
    • Method Symbol Table for Superclass
• Method Descriptors
  – Local Variable Symbol Table (Local Variable Descriptors)
    • Parameter Symbol Table (Parameter Descriptors)
      – Field Symbol Table of Receiver Class
• Local, Parameter and Field Descriptors
  – Type Descriptors in Type Symbol Table or Class Descriptors
Translating from Abstract Syntax Trees to Symbol Tables
Intermediate Representation for Classes

class vector {
    int v[];
    void add(int x) {
        int i; i = 0;
        while (i < v.length) { v[i] = v[i]+x; i = i+1; }
    }
}
class_decl

vector

field_decl

int v

method_decl

add

param_decl

int x

var_decl

int i

statements

class symbol table
class descriptor
for vector

class symbol
table

class decl

vector

field decl
int v

method decl
add param decl
int x
var decl
int i

statements
class_decl

vector field Decl

int v

method_decl

add param_decl

int x

var_decl

int i

vector class symbol table

class descriptor for vector

field descriptor
class decl
  vector field decl
    int v
  method decl
    add param decl
      int x
      int i
  var decl
    int i

class symbol table
  vector
  class descriptor for vector
    add
      Method descriptor for add
        this
          this descriptor
        i
            local descriptor

Field descriptor
  v

Parameter descriptor
  x

Statements
Intermediate Representation for Code

while (i < v.length)
    v[i] = v[i] + x;

while
    <
        sta
            +
                ldp
                    lda
                        ldf
                            ldl
                                lda
                                    ldf
                                        ldl
                                            lda
                                                ldf
                                                    ldl
                                                        ldf
                                                            ldl
                                                                lda
                                                                    ldf
                                                                        ldl
                                                                            lda
                                                                                lda
                                                                                    lda
                                                                                        len
                                                                                            ldl
                                                                                                ldf
                                                                                                       ldf
                                                                                                           ldf
                                                                                                               field descriptor for v
                                                                                                                             local descriptor for i
                                                                                                                               parameter descriptor for x
while (i < v.length)

  v[i] = v[i] + x;

field descriptor for v  local descriptor for i  parameter descriptor for x
while (i < v.length)
    v[i] = v[i] + x;
while (i < v.length)
    v[i] = v[i] + x;
while (i < v.length)
    v[i] = v[i] + x;
while (i < v.length)

v[i] = v[i]+x;
while (i < v.length)

v[i] = v[i] + x;
while (i < v.length)
  v[i] = v[i] + x;

field descriptor for v  local descriptor for i  parameter descriptor for x
while (i < v.length)
    v[i] = v[i] + x;
while (i < v.length) 

\[ v[i] = v[i] + x; \]
while (i < v.length)  
    v[i] = v[i] + x;
while (i < v.length)

\[ v[i] = v[i] + x; \]
while (i < v.length)

text

v[i] = v[i]+x;
while (i < v.length) 

\[ v[i] = v[i] + x; \]
while (i < v.length)
    v[i] = v[i] + x;

field descriptor for v  local descriptor for i  parameter descriptor for x
while (i < v.length)
    v[i] = v[i]+x;

while
    <
    sta
    len
    ldf
    lda
    ldp
    ldf
    ldl

field descriptor for v
local descriptor for i
parameter descriptor for x
while (i < v.length)

v[i] = v[i] + x;
**Parameter Descriptors**

- When build parameter descriptor, have
  - name of type
  - name of parameter

- What is the check? Must make sure 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
Local Descriptors

- When build local descriptor, have
  - name of type
  - name of local

- What is the check? Must make sure 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
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
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
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
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)
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
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
Add Rules

• Basic Principle: Hierarchy of number types (int, then float, then 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

• Interesting oddity: C converts float procedure arguments to doubles. Why?
Type Inference

- Infer types without explicit type declarations
- Add is very restricted case of type inference
- Big topic in recent programming language research
  - How many type declarations can you omit?
  - Tied to polymorphism
Equality Expressions

• If build expression $A = B$, must check compatibility
• $B$ must be compatible with $A$
  – Can always substitute a $B$ for an $A$
  – $B$ satisfies all the requirements for an $A$
  – $B$ can do at least as much as $A$
• Int compatible with Int
• Float compatible with Int, Int compatible with Float
• Class $D$ compatible with Class $C$ if $D$ inherits from $C$ (but not vice-versa)
Inheritance Example - Point Class

class point {
    int c;
    int getColor() { return(c); }
    int distance() { return(0); }
}

class cartesianPoint extends point{
    int x, y;
    int distance() { return(x*x + y*y); }
}

class polarPoint extends point {
    int r, t;
    int distance() { return(r*r); }
    int angle() { return(t); }
}
Object Interfaces

- Point - getColor(); distance();
- CartesianPoint – getColor(); distance();
- PolarPoint – getColor(); distance(); angle();
- Semantic check
  - Use type declarations
  - Check that object implements every invoked method

Point p = new PolarPoint();
p.distance(); // checks
p.angle(); // does not check
Legal and Illegal Code Sequences

• Legal code sequences
  – Point p = new Point(); p.distance();
  – Point p = new CartesianPoint(); p.distance();
  – PolarPoint o = new PolarPoint(); o.angle();
  – Point p; PolarPoint o; p = o;

• Illegal code sequences
  – Point p = new PolarPoint(); p.angle();
  – Point p; PolarPoint o; o = p;
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
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
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?
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