

# Winter 2007-2008 Compiler Construction T8 – semantic analysis recap + IR part 1

Mooly Sagiv and Roman Manevich  
School of Computer Science  
Tel-Aviv University

## Announcements

- נא להשתתף בסקר ההוראה
- Take grader's comments to your attention
  - You must document your code with javadoc
  - You must create a testing strategy
  - Catch all exceptions you throw in main and handle them (don't re-throw)
  - Print a sensible error message
- Compiler usage format:  
`java IC.Compiler <file.ic> [options]`
  - Check that there is a file argument, don't just crash
- Don't do semantic analysis in IC.cup

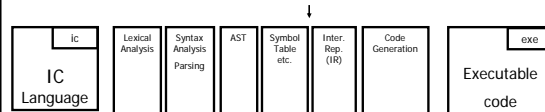
2

## Announcements

- What is expected in PA3 documentation (5 pts)
  - Testing strategy: lists of tests and what they check (scope rules, type rules etc.)
  - How did you conduct the semantic analysis – in which order do you do things
  - Don't document the most obvious things ("The input to CUP is in IC.cup"), don't repeat stuff already in PA3.pdf
  - Document non-obvious choices ("We build the type table *after* parsing", "we do not handle situation X", "we handle special situation Y like this...")
  - Output format (e.g., "first we print type table then we print...")
- Know thy group's code

3

## Today



- Today:
  - Semantic analysis recap (PA3)
  - Intermediate representation (PA4)
    - (HIR and) LIR
    - Beginning IR lowering

4

## Semantic analysis flow example

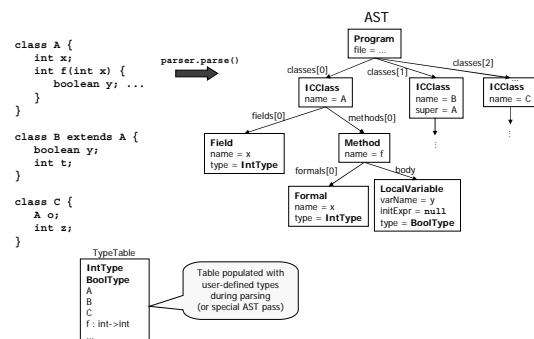
```
class A {
  int x;
  int f(int x) {
    boolean y; ...
  }
}

class B extends A {
  boolean y;
  int t;
}

class C {
  A o;
  int z;
}
```

5

## Parsing and AST construction



6

## Defined types and type table

```
class A {
  int x;
  int f(int x) {
    boolean y; ...
  }
}
```

```
class B extends A {
  boolean y;
  int t;
}
```

```
class C {
  A o;
  int z;
}
```

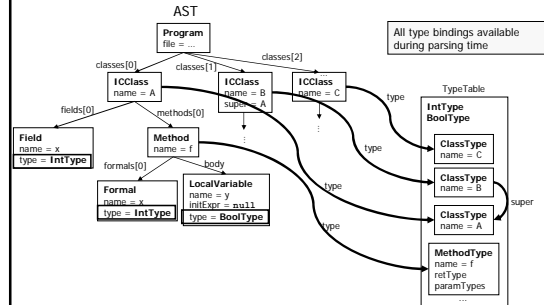
```
TypeTable
IntType
BoolType
A
B
C
f: Int->Int
...
```

```
abstract class Type {
  String name;
  boolean subtypeOf(Type t) {...}
}
class IntType extends Type {...}
class BoolType extends Type {...}
class ArrayType extends Type {
  Type elementType;
}
class MethodType extends Type {
  Type[] paramTypes;
  Type returnType;
}
class ClassType extends Type {
  ICClass classAST;
}
class ICClass extends Type {
  ICClass classAST;
}
```

```
class TypeTable {
  public static Type boolType = new BoolType();
  public static Type intType = new IntType();
  ...
  public static ArrayType arrayType(Type elementType) {...}
  public static ClassType classType(String name, String super,
    ICClass ast) {...}
  public static MethodType methodType(String name, Type returnType,
    Type[] paramTypes) {...}
}
```

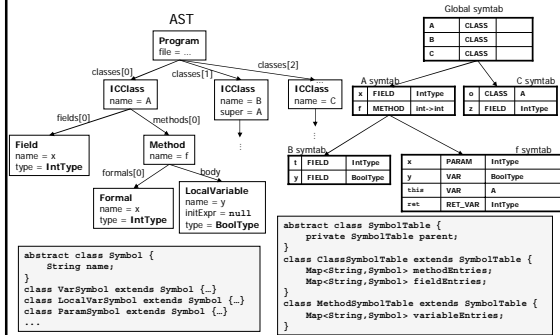
7

## Assigning types by declarations



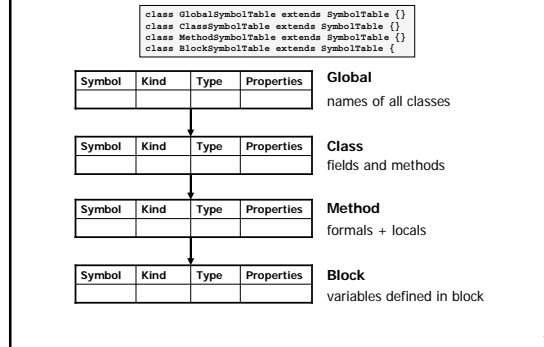
8

## Symbol tables



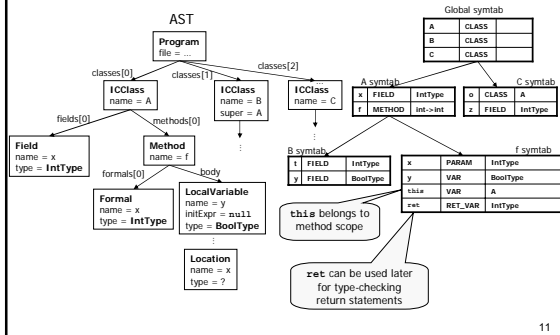
10

## Scope nesting in IC



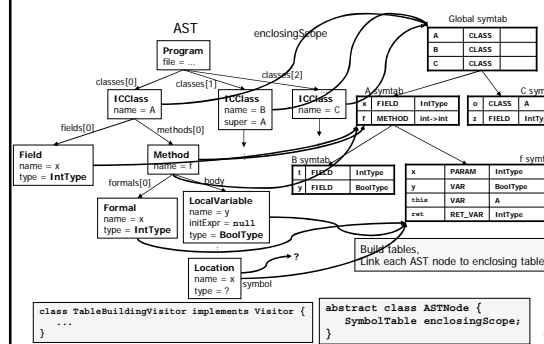
10

## Symbol tables



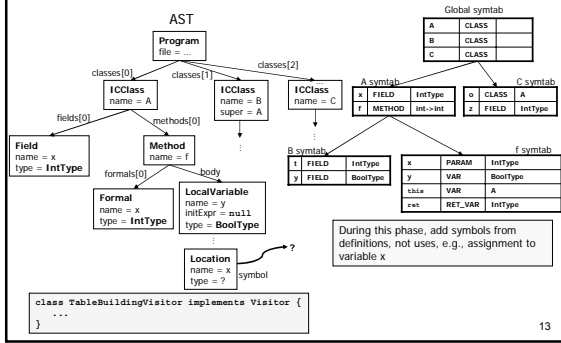
11

## Sym. tables phase 1: construct



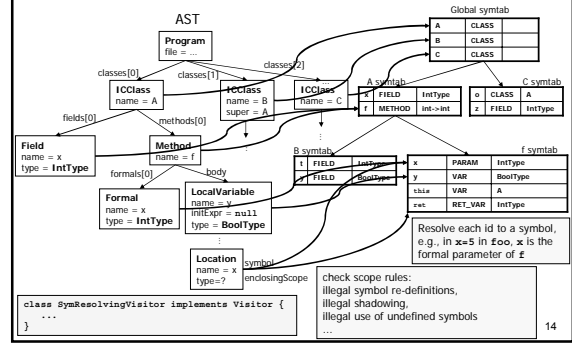
12

## Sym. tables phase 1 : construct



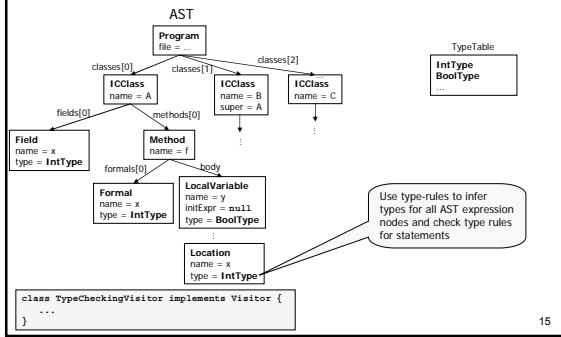
13

## Sym. tables phase 2 : resolve



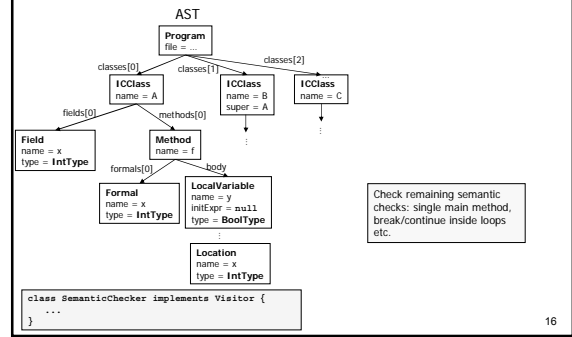
14

## Type-check AST



15

## Miscellaneous semantic checks



16

## How to write PA3

1. Implement skeleton of type hierarchy + type table
  1. Modify IC.cup/Library.cup to use types
  2. Check result using `-print-ast` option
2. Implement **Symbol** classes and **SymbolTable** classes
3. Implement symbol table construction
  - Check using `-dump-symtab` option
4. Implement symbol resolution
5. Implement checks
  1. Scope rules
  2. Type-checks
  3. Remaining semantic rules

17

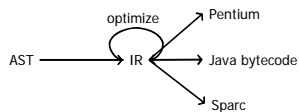
## Class quiz

- Classify the following events according to compile time / runtime / other time (Specify exact compiler phase and information used by the compiler)
  1. `x` is declared twice in method `f00`
  2. Grammar `G` is ambiguous
  3. Attempt to dereference a null pointer `x`
  4. Number of arguments passed to `f00` is different from number of parameters
  5. reduce/reduce conflict between two rules
  6. Assignment to `a+5` is illegal since it is not an l-value
  7. The non-terminal `X` does not derive any finite string
  8. Test expression in an `if` statement is not Boolean
  9. `$x` is not a legal class name
  10. Size of the activation record for method `f00` is 40 bytes

18

## Intermediate representation

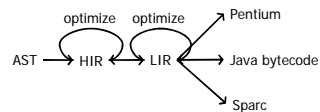
- Allows language-independent, machine independent optimizations and transformations
- Easy to translate from AST
- Easy to translate to assembly
- Narrow interface: small number of node types (instructions)



19

## Multiple IRs

- Some optimizations require high-level structure
- Others more appropriate on low-level code
- Solution: use multiple IR stages



20

## What's in an AST?

- Administration
  - Declarations
    - For example, class declarations
    - Many nodes do not generate code
- Expressions
  - Data manipulation
- Flow of control
  - If-then, while, switch
  - Target language (usually) more limited
    - Usually only jumps and conditional jumps

21

## High-level IR (HIR)

- Close to AST representation
  - High-level language constructs
- Statement and expression nodes
  - Method bodies
  - Only program's computation
- Statement nodes
  - if nodes
  - while nodes
  - statement blocks
  - assignments
  - break, continue
  - method call and return

22

## High-level IR (HIR)

- Expression nodes
  - unary and binary expressions
  - Array accesses
  - field accesses
  - variables
  - method calls
  - New constructor expressions
  - length-of expressions
  - Constants

In this project we have HIR=AST

23

## Low-level IR (LIR)

- An abstract machine language
  - Generic instruction set
  - Not specific to a particular machine
- Low-level language constructs
  - No looping structures, only labels + jumps/conditional jumps
- We will use – two-operand instructions
  - Advantage – close to Intel assembly language
  - LIR spec, available on web-site
  - Will be used in PA4
- Other alternatives
  - Three-address code:  $a = b \text{ OP } c$ 
    - Has at most three addresses (or fewer)
    - Also named quadruples:  $(a,b,c,OP)$
  - Stack machine (Java bytecodes)

24

## Arithmetic / logic instructions

- Abstract machine supports variety of operations
  - $a = b \text{ OP } c$        $a = \text{OP } b$
- Arithmetic operations: ADD, SUB, DIV, MUL
- Logic operations: AND, OR
- Comparisons: EQ, NEQ, LEQ, GE, GEQ
- Unary operations: MINUS, NEG

25

## Data movement

- Copy instruction:  $a = b$
- Load/store instructions:
  - $a = *b$        $*a = b$
- Address of instruction  $a = \&b$ 
  - Not used by IC
- Array accesses:
  - $a = b[i]$        $a[i] = b$
- Field accesses:
  - $a = b.f$        $a.f = b$

26

## Branch instructions

- Label instruction
  - label L
- Unconditional jump: go to statement after label L
  - jump L
- Conditional jump: test condition variable a; if true, jump to label L
  - cjump a L
- Alternative: two conditional jumps:
  - tjump a L      fjump a L

27

## Call instruction

- Supports call statements
  - call f(a1,...,an)
- And function call assignments
  - $a = \text{call } f(a1, \dots, an)$
- No explicit representation of argument passing, stack frame setup, etc.

28

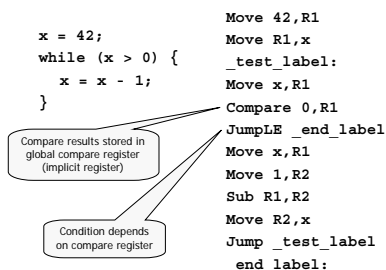
## LIR instructions

Instruction	Meaning
Move c,Rn	$Rn = c$ <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">Immediate (constant)</span>
Move x,Rn	$Rn = x$ <span style="border: 1px solid black; border-radius: 50%; padding: 2px;">Memory (variable)</span>
Move Rn,x	$x = Rn$
Add Rm,Rn	$Rn = Rn + Rm$
Sub Rm,Rn	$Rn = Rn - Rm$
Mul Rm,Rn	$Rn = Rn * Rm$
	...

Note 1: rightmost operand = operation destination  
 Note 2: two register instr - second operand doubles as source and destination

29

## Example



(warning: code shown is a naive translation)

30

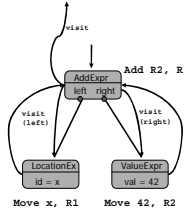
## Translation (IR Lowering)

- How to translate HIR to LIR?
- Assuming HIR has AST form (ignore non-computation nodes)
  - Define how each HIR node is translated
  - Recursively translate HIR (HIR tree traversal)
- $TR[e]$  = LIR translation of HIR construct  $e$ 
  - A sequence of LIR instructions
  - Temporary variables = new locations
    - Use temporary variables (LIR registers) to store intermediate values during translation

31

## Translating expressions – example

$TR[x + 42]$



Move x, R1  
Move 42, R2  
Add R2, R1

Very inefficient translation – can we do better?

32

## Translating expressions

- (HIR) AST Visitor
  - Generate LIR sequence for each visited node
  - Propagating visitor – register information
- When visiting an expression node
  - A single Target register designated for storing result
  - A set of available auxiliary registers
  - $TR[\text{node}, \text{target}, \text{available set}]$
- Leaf nodes
  - Emit code using target register
  - No auxiliaries required
- What about internal nodes?

33

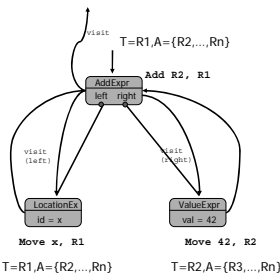
## Translating expressions

- Internal nodes
  - Process first child, store result in target register
  - Process second child
    - Target is now occupied by first result
    - Allocate a new register Target2 from available set for result of second child
  - Apply node operation on Target and Target2
  - Store result in Target
  - All initially available register now available again
  - Result of internal node stored in Target (as expected)

34

## Translating expressions – example

$TR[x + 42, T, A]$



Move x, R1  
Move 42, R2  
Add R2, R1

35

See you next week

36