

Compiler Construction

Semantic Analysis II

Rina Zviel-Girshin and Ohad Shacham
School of Computer Science
Tel-Aviv University

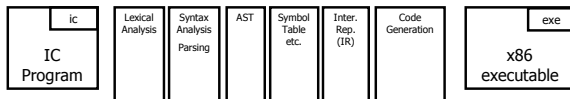
Administration

- TA1 is up
 - LR parsing
 - Submission deadline 20/12/2009
- PA3 is up
 - Submission deadline 16/12/2009

2

IC compiler

Compiler



We saw:

- Scope
- Symbol tables

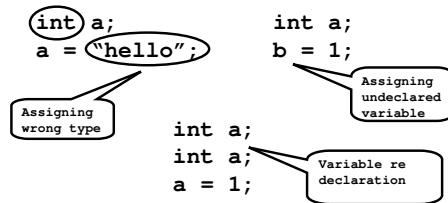
Today:

- Type checking
- Recap

3

Semantic analysis motivation

Syntax analysis is not enough



4

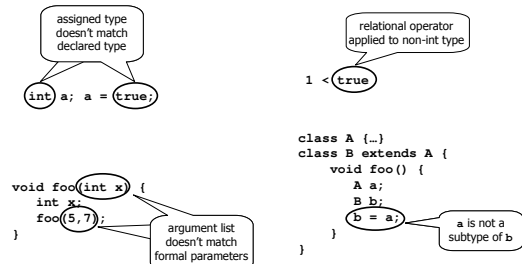
Symbol table

- An environment that stores information about identifiers
- A data structure that captures scope information

Symbol	Kind	Type	Properties
value	field	int	...
test	method	-> int	private
setValue	method	int -> void	public

5

Examples of type errors



6

Types

- Type
 - Set of values computed during program execution
 - `boolean = {true, false}`
 - `int = {-231..231-1}`
 - `void = {}`
- Type safety
 - Types usage adheres formally defined typing rules

7

Type judgments

- $e : T$
 - Formal notation for type judgments
 - e is a well-typed expression of type T
 - $2 : \text{int}$
 - $2 * (3 + 4) : \text{int}$
 - $\text{true} : \text{bool}$
 - $\text{"Hello"} : \text{string}$

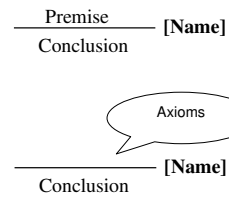
8

Type judgments

- $E \vdash e : T$
 - Formal notation for type judgments
 - In the context E , e is a well-typed expression of T
 - $b:\text{bool}, x:\text{int} \vdash b:\text{bool}$
 - $x:\text{int} \vdash 1 + x < 4:\text{bool}$
 - $\text{foo}:\text{int} \rightarrow \text{string}, x:\text{int} \vdash \text{foo}(x) : \text{string}$
- Type context
 - set of type bindings $\text{id} : T$ (symbol table)

9

Typing rules



10

Typing rules for expressions

$$\frac{E \vdash e_1 : \text{int} \quad E \vdash e_2 : \text{int}}{E \vdash e_1 + e_2 : \text{int}} \text{ [+]}$$

AST leaves

$$\frac{}{E \vdash \text{true} : \text{bool}}$$

$$\frac{}{E \vdash \text{false} : \text{bool}}$$

$$\frac{}{E \vdash \text{int-literal} : \text{int}}$$

$$\frac{}{E \vdash \text{string-literal} : \text{string}}$$

$$\frac{}{E \vdash \text{null} : \text{null}}$$

$$\frac{}{E \vdash \text{newT}() : T}$$

11

Some IC expression rules 1

$$\frac{}{E \vdash \text{true} : \text{bool}}$$

$$\frac{}{E \vdash \text{false} : \text{bool}}$$

$$\frac{}{E \vdash \text{int-literal} : \text{int}}$$

$$\frac{}{E \vdash \text{string-literal} : \text{string}}$$

$$\frac{E \vdash e_1 : \text{int} \quad E \vdash e_2 : \text{int}}{E \vdash e_1 \text{ op } e_2 : \text{int}} \quad \text{op} \in \{ +, -, /, *, \% \}$$

$$\frac{E \vdash e_1 : \text{int} \quad E \vdash e_2 : \text{int}}{E \vdash e_1 \text{ rop } e_2 : \text{bool}} \quad \text{rop} \in \{ <=, <, >, >= \}$$

$$\frac{E \vdash e_1 : T \quad E \vdash e_2 : T}{E \vdash e_1 \text{ rop } e_2 : \text{bool}} \quad \text{rop} \in \{ =, != \}$$

12

Some IC expression rules 2

$$\frac{E \vdash e1 : \text{bool} \quad E \vdash e2 : \text{bool}}{E \vdash e1 \text{ \&\& } e2 : \text{bool}} \quad \text{lop} \in \{ \&\&, || \}$$

$$\frac{E \vdash e1 : \text{int}}{E \vdash -e1 : \text{int}} \quad \frac{E \vdash e1 : \text{bool}}{E \vdash !e1 : \text{bool}}$$

$$\frac{E \vdash e1 : T[]}{E \vdash e1.length : \text{int}} \quad \frac{E \vdash e1 : T[] \quad E \vdash e2 : \text{int}}{E \vdash e1[e2] : T} \quad \frac{E \vdash e1 : \text{int}}{E \vdash \text{new } T[e1] : T[]}$$

$$\frac{}{E \vdash \text{new } T() : T} \quad \frac{E \vdash e:C \quad (\text{id} : T) \in C}{E \vdash e.\text{id} : T}$$

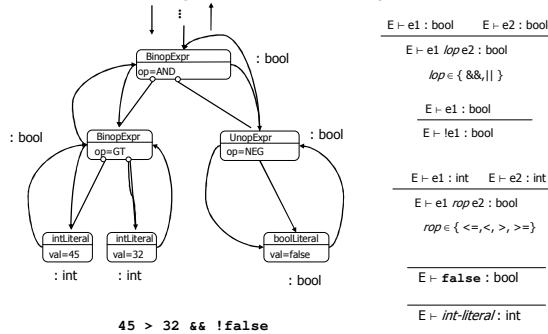
13

Type-checking algorithm

1. Construct types
 1. Add basic types to type table
 2. Traverse AST looking for user-defined types (classes, methods, arrays) and store in table
 3. Bind all symbols to types
2. Traverse AST bottom-up (using visitor)
 1. For each AST node find corresponding rule (there is only one for each kind of node)
 2. Check if rule holds
 1. **Yes**: assign type to node according to consequent
 2. **No**: report error

14

Algorithm example



15

Statement rules

- Statements have type **void**
- Judgments of the form $E \vdash S$
 - In environment E, S is well-typed

$$\frac{E \vdash e:\text{bool} \quad E \vdash S}{E \vdash \text{while}(e) S} \quad \frac{E \vdash e:\text{bool} \quad E \vdash S}{E \vdash \text{if}(e) S} \quad \frac{E \vdash e:\text{bool} \quad E \vdash S_1 \quad E \vdash S_2}{E \vdash \text{if}(e) S_1 \text{ else } S_2}$$

$$\frac{}{E \vdash \text{break}}$$

$$\frac{}{E \vdash \text{continue}}$$

16

Checking return statements

- Special entry $\{\text{ret}:T_r\}$ represents return value
 - Add to symbol table when entering method
 - Lookup entry when hit return statement

$$\frac{E \vdash e:T \quad \text{ret}:T' \in E \quad T \text{ subtype of } T'}{E \vdash \text{return } e;} \quad \frac{}{\text{ret}:\text{void} \in E} \quad \frac{}{E \vdash \text{return};}$$

17

Subtyping

- Inheritance induces subtyping relation
 - Type hierarchy is a tree
 - Subtyping rules:

$$\frac{A \text{ extends } B \{...\}}{A \leq B} \quad \frac{}{A \leq A} \quad \frac{A \leq B \quad B \leq C}{A \leq C} \quad \frac{}{\text{null} \leq A}$$

- Subtyping does not extend to array types
 - A subtype of B then A[] is not a subtype of B[]

18

Type checking with subtyping

- $S \leq T$
 - S may be used whenever T is expected
 - An Expression E from type S also has type T

$$\frac{E \vdash e : S \quad S \leq T}{E \vdash e : T}$$

19

IC rules with subtyping

$$\frac{E \vdash e1 : T1 \quad E \vdash e2 : T2 \quad T1 \leq T2 \text{ or } T2 \leq T1 \quad \text{op} \{=, \neq\}}{E \vdash e1 \text{ op } e2 : \text{bool}}$$

20

Semantic analysis flow

- Parsing and AST construction
 - Combine library AST with IC program AST
- Construct and initialize global type table
- Phase 1: Symbol table construction
 - Construct class hierarchy and check that hierarchy is a tree
 - Construct remaining symbol table hierarchy
 - Assign enclosing-scope for each AST node
- Phase 2: Scope checking
 - Resolve names
 - Check scope rules using symbol table
- Phase 3: Type checking
 - Assign type for each AST node
- Phase 4: Remaining semantic checks

21

Class hierarchy for types

```
abstract class Type {...}

class IntType extends Type {...}

class BoolType extends Type {...}

class ArrayType extends Type {
    Type elemType;
}

class MethodType extends Type {
    Type[] paramTypes;
    Type returnType;
    ...
}

class ClassType extends Type {
    ICClass classAST;
    ...
}
```

22

Type comparison

- Use a unique object for each distinct type
 - Resolve each type expression to same object
 - Use reference equality for comparison (==)

23

Type table implementation

```
class TypeTable {
    // Maps element types to array types
    private Map<Type, ArrayType> uniqueArrayTypes;
    private Map<String, ClassType> uniqueClassTypes;

    public static Type boolType = new BoolType();
    public static Type intType = new IntType();
    ...

    // Returns unique array type object
    public static ArrayType arrayType(Type elemType) {
        if (uniqueArrayTypes.containsKey(elemType)) {
            // array type object already created - return it
            return uniqueArrayTypes.get(elemType);
        }
        else {
            // object doesn't exist - create and return it
            ArrayType arrt = new ArrayType(elemType);
            uniqueArrayTypes.put(elemType, ArrayType);
            return arrt;
        }
    }
    ...
}
```

24

Recap

Semantic analysis flow example

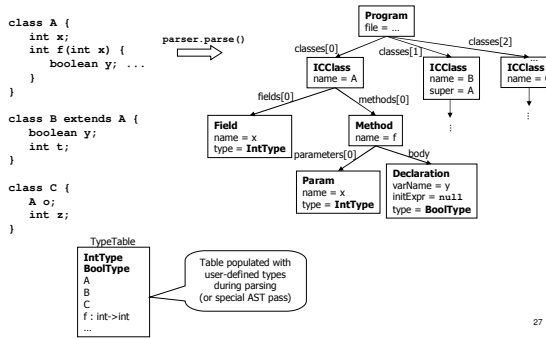
```

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

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

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

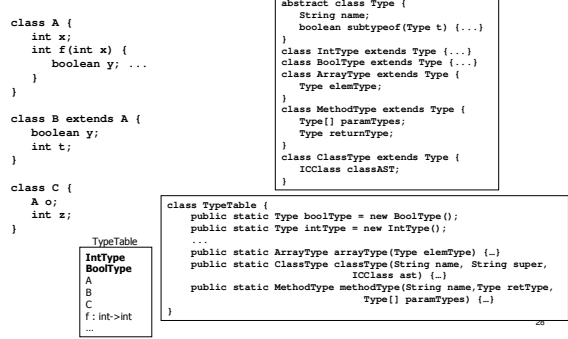
Parsing and AST construction



TypeTable
 IntType
 BoolType
 A
 B
 C
 f: int->int
 ...

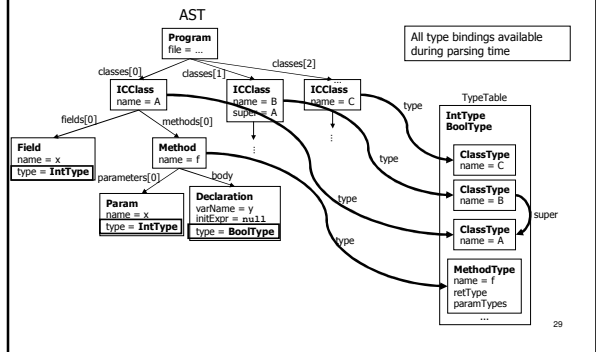
Table populated with user-defined types during parsing (or special AST pass)

Defined types and type table



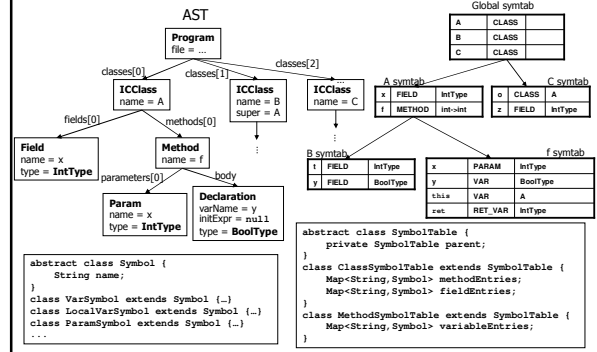
TypeTable
 IntType
 BoolType
 A
 B
 C
 f: int->int
 ...

Assigning types by declarations



All type bindings available during parsing time

Symbol tables



```

abstract class SymbolTable {
    private SymbolTable parent;
    String name;
}

class ClassSymbolTable extends SymbolTable {
    Map<String, Symbol> methodEntries;
    Map<String, Symbol> fieldEntries;
}

class LocalVarSymbol extends Symbol { ... }

class ParamSymbol extends Symbol { ... }
    
```

Scope nesting in IC

```

class GlobalSymbolTable extends SymbolTable {}
class ClassSymbolTable extends SymbolTable {}
class MethodSymbolTable extends SymbolTable {}
class BlockSymbolTable extends SymbolTable {}

```

Symbol	Kind	Type	Properties	Global
				names of all classes

Symbol	Kind	Type	Properties	Class
				fields and methods

Symbol	Kind	Type	Properties	Method
				formals + locals

Symbol	Kind	Type	Properties	Block
				variables defined in block

31

Symbol tables

32

Sym. tables phase 1 : construction

```

class TableBuildingVisitor implements Visitor {
    ...
}

```

33

Sym. tables phase 1 : construct

34

Sym. tables phase 2 : resolve

```

class SymResolvingVisitor implements Visitor {
    ...
}

```

35

Type-check AST

```

class TypeCheckingVisitor implements Visitor {
    ...
}

```

36

Miscellaneous semantic checks

