COMP3131/9102: Programming Languages and Compilers

Jingling Xue

School of Computer Science and Engineering
The University of New South Wales
Sydney, NSW 2052, Australia

`http://www.cse.unsw.edu.au/~cs3131`

`http://www.cse.unsw.edu.au/~cs9102`

Lectures (Schedule)

1. Week 1: Intro, lexical analysis, DFAs and NFAs √
2. Week 2: CFGs + parsing √
3. Week 3: Abstract syntax trees (ASTs) √
4. Week 4: Attribute grammars √
5. Week 5: Static semantics
6. Week 6: Half-term break
7. Week 7: Jasmin
8. Week 8: Code generation
9. Week 9: DFAs + NFAs + Parsing
10. Week 10: Revision

Lecture 8: Static Semantics

The semantic analyser enforces a language's semantic constraints

1. Two types of semantic constraints:
   - Scope rules
   - Type rules
2. Two subphases in semantic analysis:
   - Identification (symbol table)
   - Type checking
3. Standard environment
4. Assignment 4:
   - The visitor design pattern
   - The two subphases combined in one pass only

This week's lectures + Assn 4 spec $\Rightarrow$ Type Checker

Type Checking

- Data type: set of values plus set of operations on the values
- Typical checks:
  - Type checks: expressions well typed using the type rules
  - Flow-of-control checks: break & continue contained in a loop, etc.
  - Uniqueness checks: The labels in a switch are distinct, etc.
- Type rules: the rules to infer the type of each language construct and decide whether the construct has a valid type
- Type checking: applying the language's type rules to infer the type of each construct and comparing that type with the expected type

Type Checking in VC

- VC is statically type checked
- Lack of structured types $\Rightarrow$ simple checks:
  - Expressions: an operator applied to compatible operands
  - Statements:
    * break and continue must be contained in a loop
    * The type of the expression in a return statement in a function must be assignment-compatible with the result type of the function
    * unreachable statements
    * Every function whose result type is int or float must have a return statement (optional)

Type Checking of Expressions in VC

- The type rules are defined informally in the document:
  VC Language Definition
- Essentially, one checks if an operator is applied to compatible operands
  - The type of an unary operator: $T_1 \rightarrow T_2$
  - The type of a binary operator (including "="): $T_1 \times T_2 \rightarrow T_3$
  - The type of the function int f(int i, float f) is: int $\times$ float $\rightarrow$ int
- The compiler infers that expression E has some type T or that E is ill-typed. If E occurs in a context where T' is expected, then the compiler checks if T is equivalent to T'

The Synthesised Attributed type in Expr.java

- The abstract class Expr.java:

```
package VC.ASTs;
import VC.Scanner.SourcePosition;
public abstract class Expr extends AST {
  public Type type;
  public Expr (SourcePosition Position) {
    super (Position);
    type = null;
  }
}
```

- All concrete expr classes inherit the instance variable type

The Synthesised Attributed type

- The abstract class Var.java:

```
package VC.ASTs;
import VC.Scanner.SourcePosition;
public abstract class Var extends AST {
  public Type type;
  public Var (SourcePosition Position) {
    super (Position);
    type = null;
  }
}
```

- Both attributes inherited in the concrete class SimpleVar.java

Bottom-Up Computation of type in an Expression AST

- Literal: its type is immediately known
- Identifier: is type obtained from the inherited attribute Decl associated with every Ident node
- Binary operator application: Consider $E_1 O E_2$, where O is a binary operator of type $T_1 \times T_2 \rightarrow T_3$. The type checker ensures that E_1's type is equivalent to T_1, and E_2's type is equivalent to T_2, and thus infers that the type of $E_1 O E_2$ is T_3. Otherwise, there is a type error.
- Other operator applications dealt with similarly

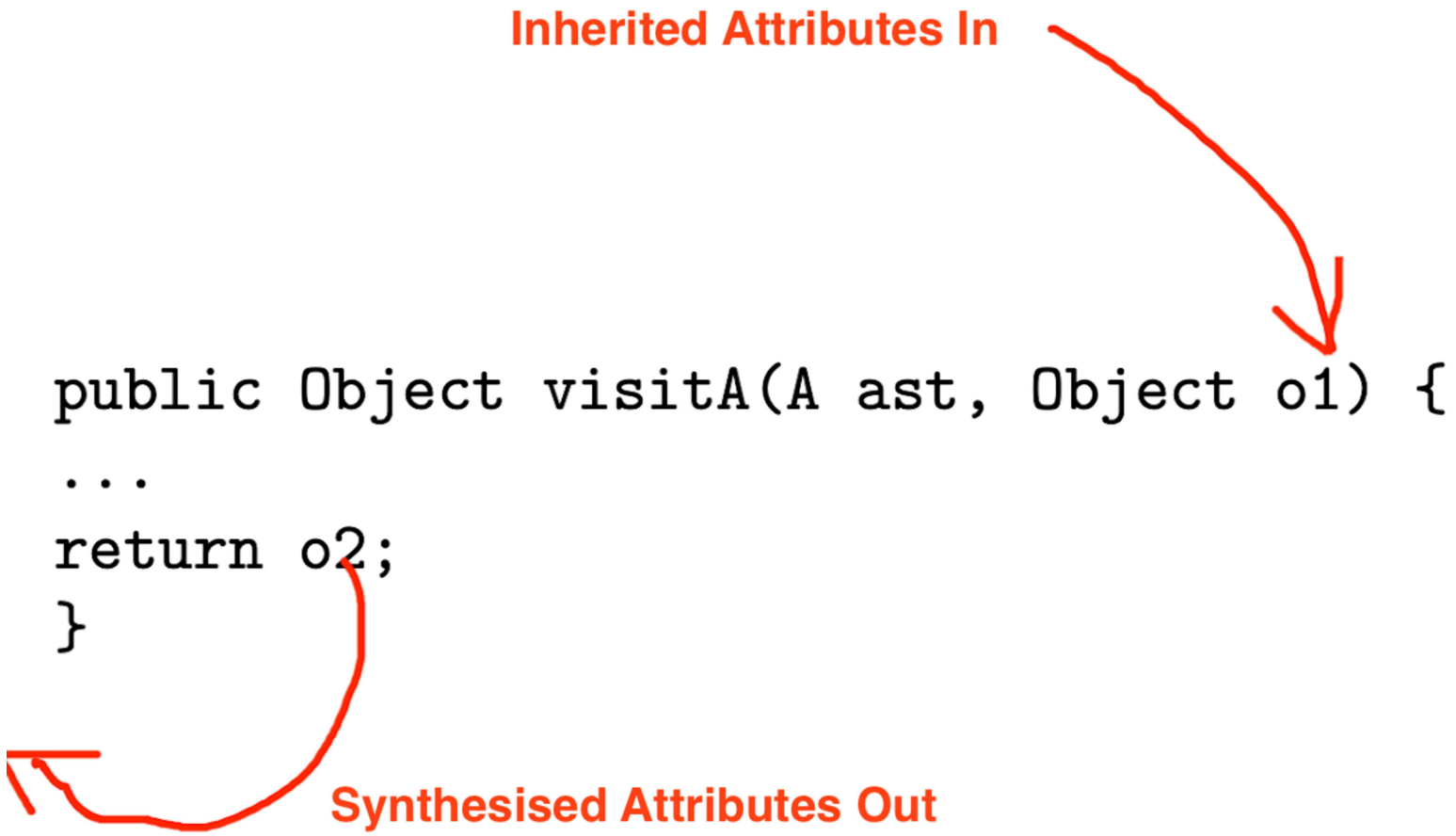
Standard Environment

- StdEnvironment is a class with the five static variables:

```
StdEnvironment.intType = new IntType(dummyPos);
StdEnvironment.floatType = new FloatType(dummyPos);
StdEnvironment.booleanType= new BooleanType(dummyPos);
StdEnvironment.stringType= new stringType(dummyPos);
StdEnvironment.voidType = new VoidType(dummyPos);
StdEnvironment.errorType = new ErrorType(dummyPos);
```

- errorType can be assigned to an ill-typed expression whose type cannot be deduced since some of its subexpressions are ill-typed

The Visit Design Pattern



Types of Identifiers and Literals

```
public Object visitIdent(Ident I, Object o) {
  Decl binding = idTable.retrieve(I.spelling);
  if (binding != null)
    I.decl = binding;
  return binding;
}
public Object visitIntLiteral(IntLiteral IL, Object o) {
  return StdEnvironment.intType;
}
public Object visitFloatLiteral(FloatLiteral IL, Object o) {
  return StdEnvironment.floatType;
}
public Object visitBooleanLiteral(BooleanLiteral IL, Object o) {
  return StdEnvironment.booleanType;
}
public Object visitStringLiteral(StringLiteral SL, Object o) {
  return StdEnvironment.stringType;
}
```

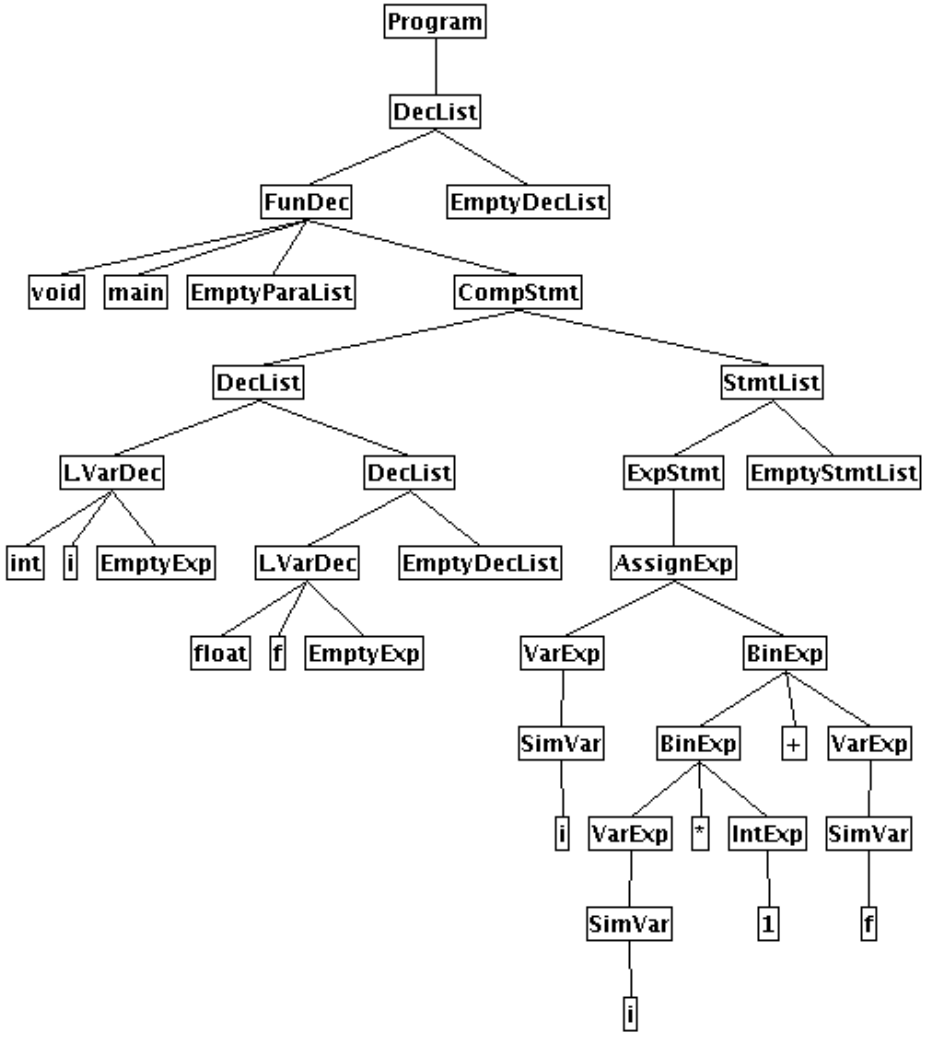
Type Coercions

- There are two types of operations at hardware level on, say, +:
  - integer addition when both operands are integers
  - floating-point addition when both operands are reals
- Type coercion: the compiler implicitly converts int to float, whenever necessary, when an expression is evaluated
- Each overloaded operator is associated two non-overloaded operators: one for integer operation and the other for floating-point operation
- Your VC compiler needs to perform two tasks:
  - Add i2f conversion operator, whenever necessary
  - Indicate if an operator is integral or real (e.g., i+ or f+)
  - See Assignment 4 spec for details

Example 4: Type Checking Expressions

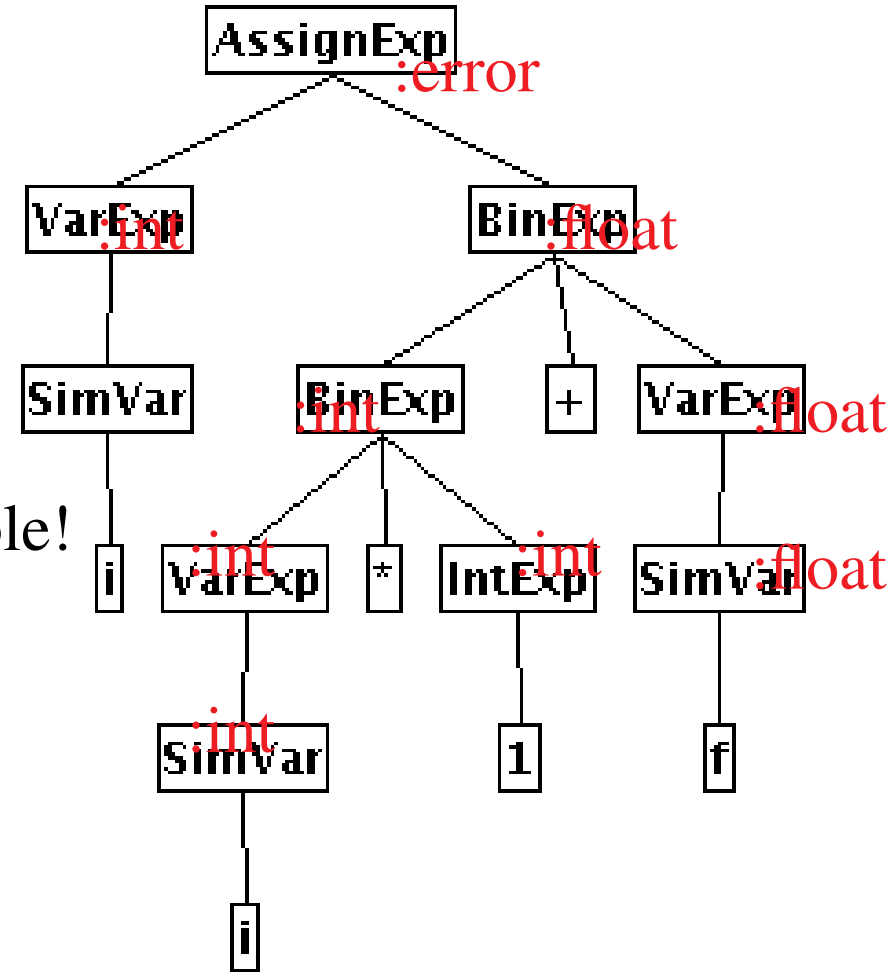
```
void main()
{
  int i;
  float f;
  i = i * 1 + f;
}
```

The AST for Example 4

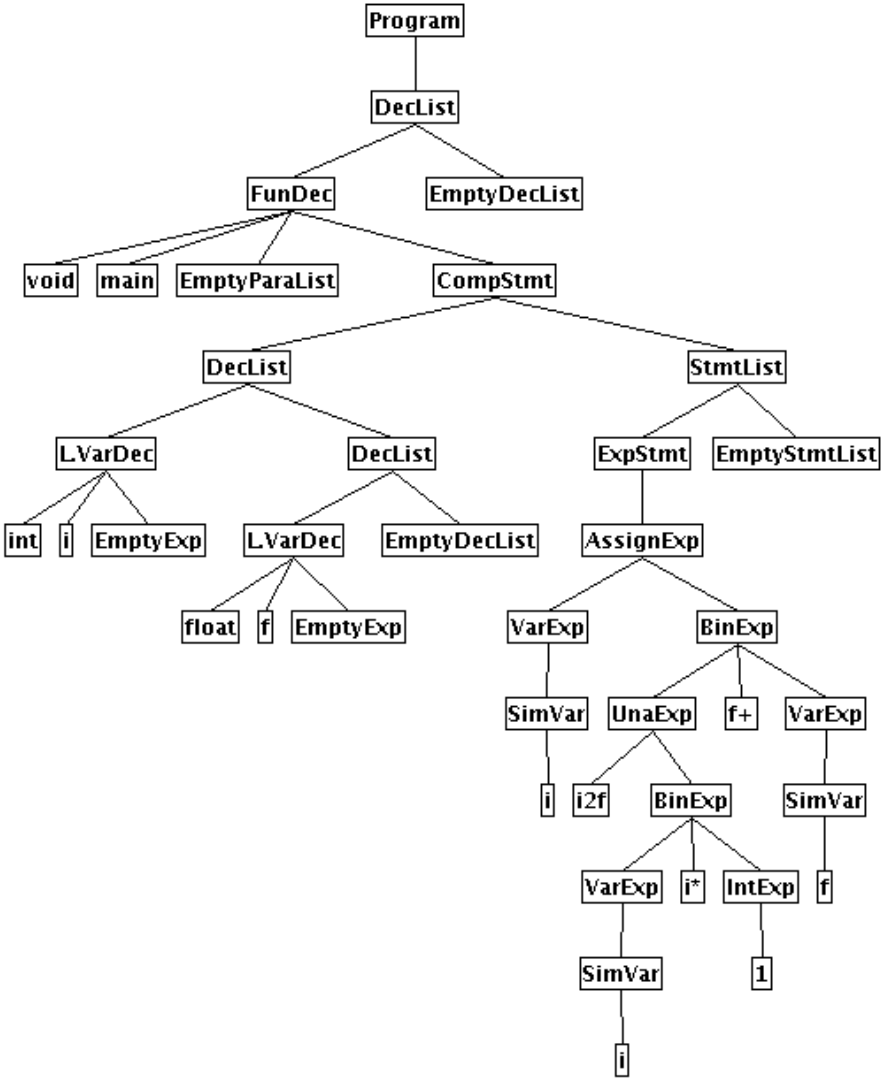


Example 4: Type Checking Expressions

Error: assignment incompatible!

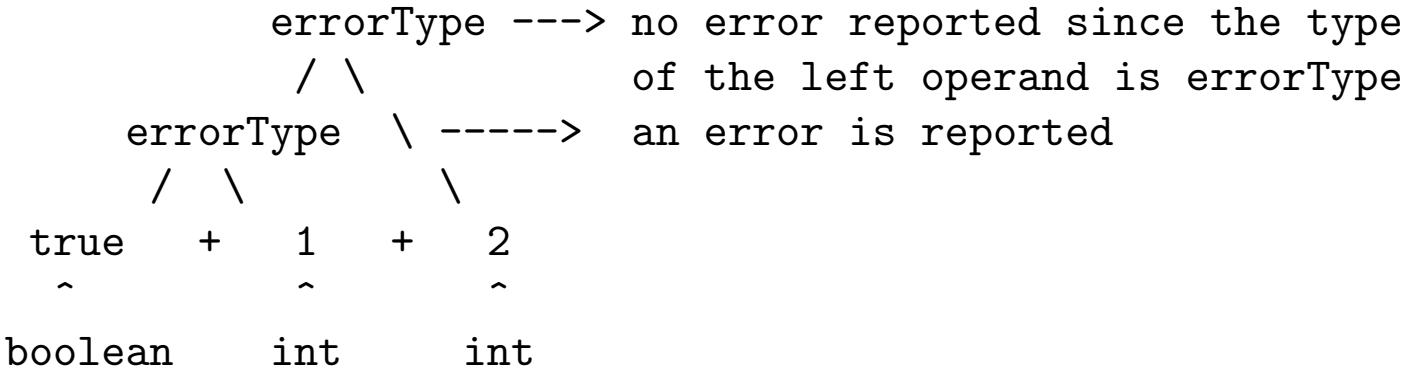


Example 4: Type Coercions (Decorated AST)



Error Detection, Reporting and Recovery: Week 8 Tutorial

- Detection: based on type rules
- reporting: prints meaningful error messages
- Recovery: continue checking types in the presence of errors:
  - Must avoid a cascade of spurious errors
  - An ill-typed expression given StdEnvironment.errorType if its type cannot be determined in the presence of errors
  - Do not report an error for an expression if any of its subexpressions is StdEnvironment.errorType



Attribute Grammar for Type Checking (Using VC's Type Rules)

Production	Semantic Rules
$\langle S \rangle \rightarrow \langle E \rangle$	$\langle S.type \rangle = \langle E.type \rangle$
$\langle E_1 \rangle \rightarrow \langle E_2 \rangle$ / $\langle E_3 \rangle$	$\langle E_1.type \rangle =$ $\begin{cases} \langle E_2.type \rangle = \text{int and } \langle E_3.type \rangle = \text{int} & \rightarrow \text{int} \\ \langle E_2.type \rangle = \text{int and } \langle E_3.type \rangle \text{ float} & \rightarrow \text{float} \\ \langle E_2.type \rangle = \text{float and } \langle E_3.type \rangle \text{ int} & \rightarrow \text{float} \\ \langle E_2.type \rangle = \text{float and } \langle E_3.type \rangle \text{ float} & \rightarrow \text{float} \\ \text{else} & \rightarrow \text{errortype} \end{cases}$
$\langle E \rangle \rightarrow$ **num**	$\langle E.type \rangle = \text{int}$
$\langle E \rangle \rightarrow$ **num . num**	$\langle E.type \rangle = \text{float}$

Assignment 4

- Implement a one-pass semantic analyser using the visitor design pattern
- Identification implemented for you
- Type checking implemented by you
  - checking
  - add i2f
  - choose a non-overloaded operation (e.g., `+=>i+ or f+`)
- Decorated ASTs:

```
The synthesised attribute, type, in Expr nodes
The synthesised attribute, type, in SimpleVar nodes
```

- The symbol table discarded once the AST is decorated

Reading

- Chapter 6 (Red Dragon) or Section 6.5 (Purple Dragon)
- TreeDrawer, TreePrinter and Unparser to understand the visitor design pattern
- On-line resources on typing if you are interested
- Assignment 4 spec
- The on-line VC language definition

Next class: JVM & Jasmin Assembly Language