COMP 520 - Compilers

Lecture 12a (Thu Mar 7)

miniJava PA3 details

- PA3 Contextual Analysis
  - due Monday Mar 25 (11:59 PM)
PA3 Project

• Implement contextual analysis in a subpackage
  – miniJava.ContextualAnalysis

• Contextual Analysis consists of
  – Identification
  – Type checking

• Also add null to miniJava!

• Sample PA3 project structure
  (within miniJava.ContextualAnalyzer
   choose classes and class names as you wish)
Implementation of ContextualAnalysis

```java
import miniJava.AbstractSyntaxTrees.*;

public class Identification implements Visitor< ?, ? > {
    ... }

public class TypeChecking implements Visitor< ?, ? > {
    ... }
```

- What type parameters should we use to replace "?"
Identification

- **How to perform identification**
  - Declarations need to be entered
    - `ClassDecl`, `MemberDecl`, `LocalDecl`
  - Identifiers need to be retrieved
    - `add field to Identifier Class: public Declaration decl`
    - work out a correct order to visit different parts of the AST to ensure all applicable declarations will have been seen before visiting an Identifier
    - link each identifier in the AST to its declaration using the appropriate idTable(s)

- **What constructs need identification?**
  - Basically all
    - Declarations
    - Statements, Expressions, References, TypeDenoters
      - anything that could contain an Identifier
Identification

- **IdTables**
  - enter(String s, Decl d)
    - associate s with Decl d
  - Decl retrieve(String s)
    - yields decl or null

- **Specific id tables**
  - is s a class name?
  - is s a member of class X?

- **Scoped id table**
  - enter or exit a scope
  - what declaration is associated with s in the current scope?
  - is s already declared in the current scope?
  - is s already declared in a scope with level ≥ 3?
  - enter a new <name,Decl> at the current scope level

<table>
<thead>
<tr>
<th>string</th>
<th>Decl</th>
<th>level</th>
</tr>
</thead>
<tbody>
<tr>
<td>class names</td>
<td>ClassDecl</td>
<td>1</td>
</tr>
<tr>
<td>member names</td>
<td>MemberDecl</td>
<td>2</td>
</tr>
<tr>
<td>parameter names</td>
<td>ParameterDecl</td>
<td>3</td>
</tr>
<tr>
<td>local var names</td>
<td>LocalDecl</td>
<td>4+</td>
</tr>
</tbody>
</table>
Identification

- **Special challenges**
  - Access and Visibility restrictions of MemberDecls
    - Non-static members are not always accessible
    - private members are not always accessible
    - need a “context” for a reference to make a judgment

- **References**
  - example
    - x.y.z
  - what needs to be checked at each node of the Reference ast?
Type Checking

- Relatively simple
  - Create a typeDenoter attribute in every Expression node (or possibly in every node)
  - The type rules for predefined functions are relatively simple
    - +, -, *, etc: Int x Int → Int
    - ==: α x α → Boolean
    - index: Array(α) x Int → α
    - assign: α x α → Stmt
  - A single upwards pass suffices for miniJava type checking

- Study the type related classes in the AST
  - TypeDenoter, TypeKind, BaseType, ArrayType, Classtype
  - create an equality function between arbitrary instances of TypeDenoter

- run only if identification has completed successfully!
  - e.g. A x = new A();
Type Checking

• Additional types
  – Error type
    • Error type is equal to any type
    • limits propagation of errors
    • gives most useful continuation of type checking after an error
  – Unsupported type
    • Unsupported type is not equal to any type
    • therefore a value of type unsupported is not type correct in any operation
    • predefined name String can have unsupported type