COMP 520 - Compilers

Lecture 12 (Tue Feb 25)

*miniJava Contextual Analysis*

- PA3 Contextual Analysis assignment is available online
  - due Monday Mar 26
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**
  - 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 at level > `k`?
  - enter a new `<name,Decl>` at the current level
  - enter or exit a scope

<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

- How to perform identification
  - Declarations need to be entered
    - ClassDecl, MemberDecl, LocalDecl
  - Identifiers need to be retrieved
    - add a Declaration attribute to the Identifier class
    - 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

- **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?
Identification

• Special challenges
  – Scope of variable in a VarDecl
    • e.g.

```
int x = 3 + x;
```

• is erroneous!

– predefined classes
  • System, etc.
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 : \( \text{Int} \times \text{Int} \rightarrow \text{Int} \)
  == : \( \alpha \times \alpha \rightarrow \text{Boolean} \)
  index : \( \text{Array}(\alpha) \times \text{Int} \rightarrow \alpha \)
  assign : \( \alpha \times \alpha \rightarrow \text{Stmt} \)

• 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

• provided identification has completed successfully!
  – e.g. \( A \times = \text{new} \ A() \);
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
Midterm

• Tuesday March 5 start 12:30 – exams collected at 1:45
  – in-class exam – 75 mins
  – Arrive early – you will be split over two classrooms so you can spread out

• Open book / notes
  – Limited to materials distributed in the current offering of COMP 520
  – You may not use a computer, phone, or other electronic devices
  – You must complete the pledge
Midterm exam scope

- **Textbook chapters 1-5**
  - Chapter 4 scanning and parsing
    - Scanning
    - EBNF context-free grammars
    - Grammar manipulation, precedence parsing
    - LL(1) property
  - Chapter 5 contextual analysis
    - Identification
    - Type checking

- **Class**
  - Lecture notes 1-12

- **Assignments**
  - WA1 – WA3
  - Project checkpoints PA1 and PA2
  - know your compiler
  - know what you will be doing in PA3