Program Representations

17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

Representing Programs

- To analyze software automatically, we must be able to represent it precisely
- Some representations
  - Source code
  - Abstract syntax trees
  - Control flow graph
  - Bytecode
  - Assembly code
  - Binary code

The WHILE Language

- A simple procedural language with:
  - assignment
  - statement sequencing
  - conditionals
  - while loops
- Used in early papers (e.g. Hoare 69) as as a “sandbox” for thinking about program semantics
- We will use it to illustrate several different kinds of analysis

WHILE Syntax

- Categories of syntax
  - S ∈ **Stmt** statements
  - a ∈ **AExp** arithmetic expressions
  - x,y ∈ **Var** variables
  - n ∈ **Num** number literals
  - b ∈ **BExp** boolean expressions
- Syntax
  - S ::= $x := a$ | skip | $S_1; S_2$
| if b then S_1 else S_2 | while b do S
  - a ::= x | n | $a_1\ op_a\ a_2$
  - op_a ::= + | - | * | / | …
  - b ::= true | false | not b | $b_1\ op_b\ b_2$ | $a_1\ op_r\ a_2$
  - op_b ::= and | or | …
  - op_r ::= $<$ | $\leq$ | = | $>$ | $\geq$ | …

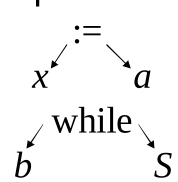
Example WHILE Program

```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```

Computes the factorial function, with the input in x and the output in z

Abstract Syntax Trees

- A tree representation of source code
- Based on the language grammar
  - One type of node for each production
  - $S ::= x := a$ ➔
  - $S ::=$ while b do S ➔



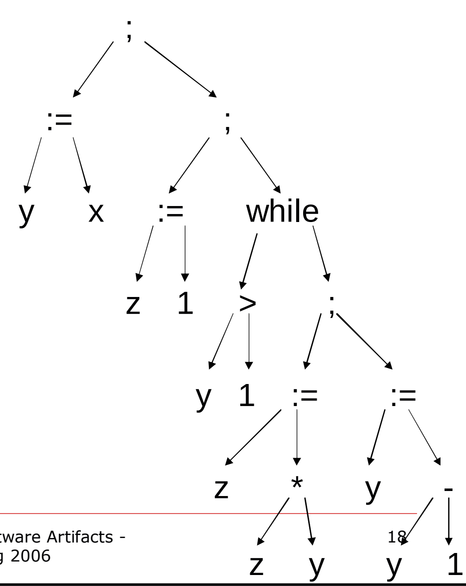
Parsing: Source to AST

- Parsing process (top down)
  1. Determine the top-level production to use
  2. Create an AST element for that production
  3. Determine what text corresponds to each child of the AST element
  4. Recursively parse each child
- Algorithms have been studied in detail
  - For this course you only need the intuition
  - Details covered in compiler courses

Analysis of Software Artifacts - Spring 2006 7

Parsing Example

y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Analysis of Software Artifacts - Spring 2006 18

WHILE ASTs in Java

- Java data structures mirror grammar
- S ::= x := a
  - | skip
  - | S_1; S_2
  - | if b then S_1 else S_2
  - | while b do S

```
class AST { ... }
class Stmt extends AST { ... }
class Assign extends Stmt {
    Var var;
    AExpr expr;
}
class Skip extends Stmt { }
class Seq extends Stmt {
    Stmt left;
    Stmt right;
}
class If extends Stmt {
    BExpr cond;
    Stmt thenStmt;
    Stmt elseStmt;
}
class While extends Stmt {
    BExpr cond;
    Stmt body;
}
```

Analysis of Software Artifacts - Spring 2006 19

Course Analysis Toolkit

- Eclipse
  - Open-source Java integrated development environment
  - Extensible through plugins
  - Exposes Java AST to plugins
- Crystal
  - Plugin for Eclipse
  - Provides a basic analysis framework
  - Supports interaction with end user

Analysis of Software Artifacts - Spring 2006 20

Extending Crystal

- Download and install Java 5
- Download and install Eclipse 3.1
- Download and install Crystal
- Implement a class that extends:
  - ICrystalAnalysis for global analyses
  - AbstractCrystalMethodAnalysis for method-at-a-time analyses
    - This will usually be the case
- Register your new analysis with Crystal
  - It can then be run from the Crystal menu

Analysis of Software Artifacts - Spring 2006 21

AbstractCrystalMethodAnalysis

- public void beforeAllMethods() { }
  - Called at the beginning of an analysis cycle
  - Use for analysis setup
- public abstract void analyzeMethod(MethodDeclaration d);
  - Invoked by the framework for each method in the system
  - You must override this to perform your analysis task for each method
- public void afterAllMethods() { }
  - Called at the end of an analysis cycle
  - Use for analysis cleanup and any reporting that's still left

Analysis of Software Artifacts - Spring 2006 22

Example: PrintMethods

```
Crystal crystal = Crystal.getInstance();
public void beforeAllMethods() {
   crystal.userOut().println("Printing methods:");
}
public void analyzeMethod(MethodDeclaration md) {
   crystal.userOut().println(md.getName());
}
public void afterAllMethods() {
   crystal.userOut().println("Done.");
}
```

Analysis of Software Artifacts - Spring 2006 23

Registering the Analysis

- In CrystalPlugin.java:

```
public void setupCrystalAnalyses(
                                   Crystal crystal) {
   PrintMethods pm = new PrintMethods();
   crystal.registerAnalysis(pm);
}
```

Analysis of Software Artifacts - Spring 2006 24

The Eclipse AST

- View Tree
- Browse javadoc for:
  - MethodDeclaration
  - Block
  - Statement
  - VariableDeclarationStatement
  - VariableDeclaration
  - ExpressionStatement
  - Assignment
  - Name
  - IVariableBinding

ASTNodes and Bindings

- ASTNode
  - The AST representation of Java source
  - There will be an ASTNode for each occurrence of a variable in the source
- Binding
  - A single canonical object representing the variable
  - Eclipse doesn't provide a way to get from the Binding to the ASTNode
    - Efficiency choice
  - Crystal provides a convenient shortcut
    - ASTNode Utilities.getASTNode(IBinding b)

Demo

- Installing Crystal
- Run Assignment 0
- Look at Assignment 0 code
- Look at Visitor
- Results of Assignment 1

The Visitor Pattern

```
class Visitor {
    // called before visit
    void preVisit(Node n) { }
    // if return true, children visited
    boolean visit(Element e) {
        return true; }
    // called after child visits
    void endVisit(Element e) {
        return true; }
    // called after visit
    void postVisit(Node n) { }

}
```

```
class Node {
    abstract void accept(Visitor
    v);
}
class Element extends Node {
    void accept(Visitor v) {
        v.preVisit(this);
        boolean c = v.visit(this);
        if (c)
            children.accept(v);
        v.endVisit(this);
        v.postVisit(this);
    }
}
```