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

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

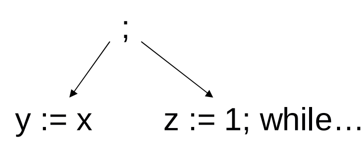
- A tree representation of source code
- Based on the language grammar
  - One type of node for each production
  - $S ::= x := a$ ➔ $:=$ with children x and a
  - $S ::=$ while b do S ➔ while with children b and S

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

Parsing Example

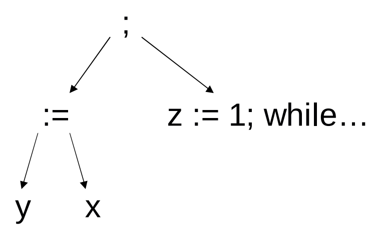
```
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1
```



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

Parsing Example

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

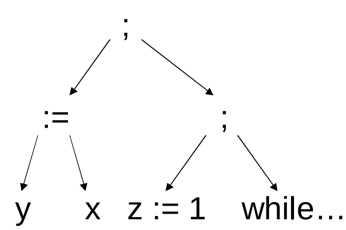


- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Analysis of Software Artifacts - Spring 2006 9

Parsing Example

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

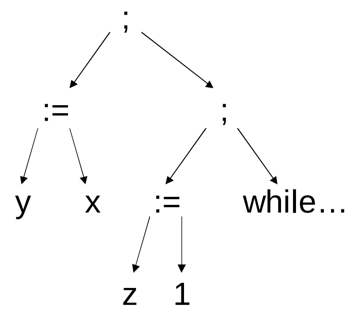


- Top-level production?
  - $S_1; S_2$
- What are the parts?
  - y := x
  - z := 1; while …

Analysis of Software Artifacts - Spring 2006 10

Parsing Example

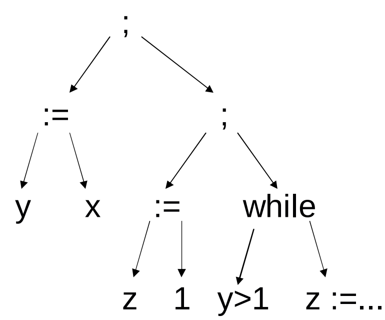
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

Parsing Example

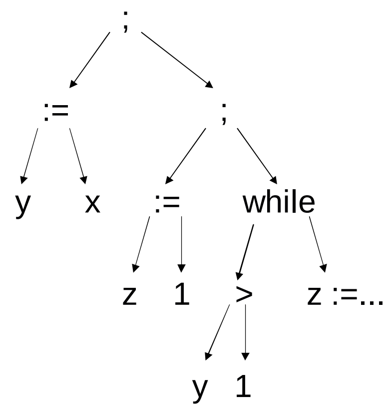
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

Parsing Example

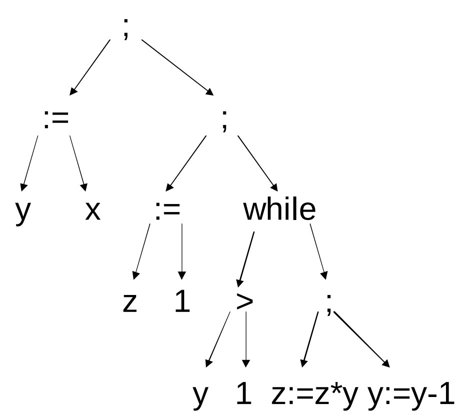
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

Parsing Example

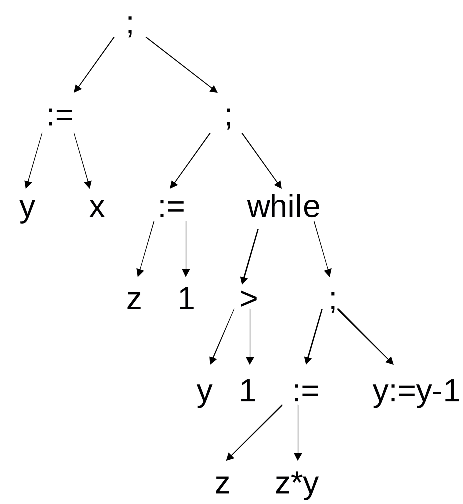
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

Parsing Example

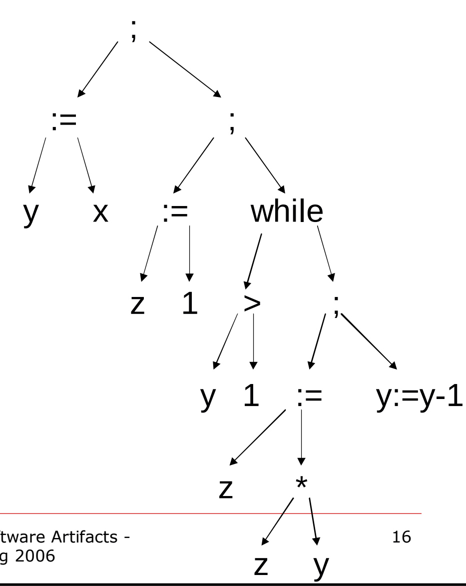
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

Parsing Example

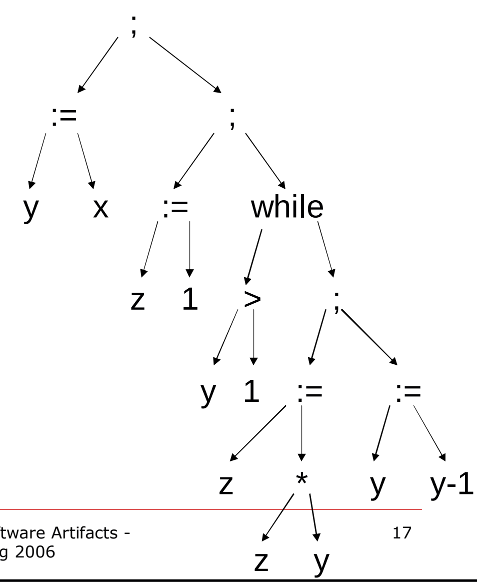
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

Parsing Example

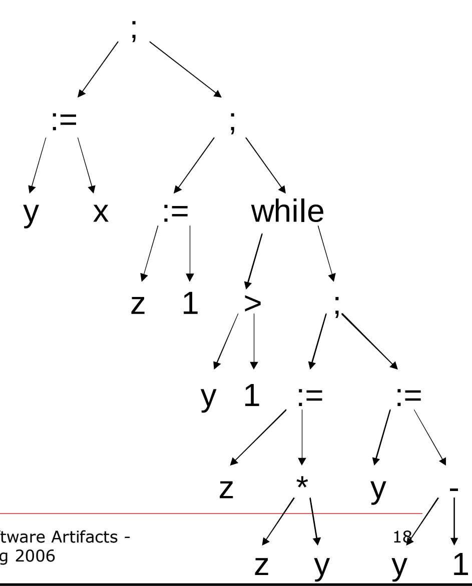
y := x;
z := 1;
while y>1 do
    z := z * y;
    y := y – 1



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

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 …

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
- Crystal
  - Plugin for Eclipse
  - Provides a Java AST for analysis
    - Eclipse has its own AST, but it's not ideal for analysis
  - Provides a basic analysis framework
  - Supports interaction with end user

Analysis of Software Artifacts - Spring 2006 20

Extending Crystal

- Download and install Java version 5 or 6
- Download and install Eclipse 3.2
- 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(IMethodDeclarationNode 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(IMethodDeclarationNode md) {
   crystal.userOut().println(md.getId());
}
public void afterAllMethods() {
   crystal.userOut().println("Done.");
}
```


Registering the Analysis

- In CrystalPlugin.java:

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

The Crystal AST

- View Tree in Eclipse
  - Interface package: com.surelogic.ast.java.operator
- Browse hierarchy
  - IJavaOperatorNode – root of tree
  - IDeclarationNode – class, field, methods
    - IClassDeclarationNode - classes
    - IMethodDeclarationNode - methods
    - IFieldDeclarationNode and DeclaratorNode – “int i,j=6,k;” vs “j=6”
  - IStatement Node
    - IBlockStatementNode – blocks { }
    - IDeclStatementNode – variables
    - If, For, Return…
    - IExprStatementNode – expression statements (incl. assignments)
  - IReturnTypeNode – void, other types
    - IPrimitiveTypeNode: boolean, int, float…
    - IReferenceTypeNode: int[], String, Foo<T>

The Crystal AST

- Browse hierarchy (continued)
  - IInitializerNode → IExpressionNode – expressions
    - IBinopExpressionNode: binary expressions
      - IArithBinopExpressionNode: + - * / etc.
      - IAssignmentExpressionNode: assignment
        - IAssignExpressionNode (regular)
        - IOpAssignExpressionNode (+=, -=, *=, etc.)
    - IUnopExpressionNode: !b, x++
    - ILiteralExpressionNode: 5, 7.3, true, “hello”, null
    - IPrimaryExpressionNode
      - IAllocationExpressionNode: new X()
      - IFieldRefNode: this.f
      - ISomeFunctionCallNode: x.m(5)
      - ISomeThisExpression: this, X.this
      - ITypeExpressionNode: used as the “receiver” of static methods (artificial)
      - IVariableUseExpressionNode: x, y
  - IBinding – binds a node to a declaration
    - example: call.resolveBinding(), type.resolveType()
    - IDeclarationNode: declarations are bindings (getNode() returns this)
    - IFieldDeclarationNode
    - IFunctionBinding: gets the function declaration
    - IType
      - IPrimitiveType – binds to IPrimitiveTypeNode
      - IDerivedRefType – arrays, etc.
      - IDeclaredType – a class or interface
      - ITypeFormal – a type parameter

Demo

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

The Visitor Pattern

```
interface IVisitor<T> {
      // one for each element type
      T visit(Element e);
}

class DescendingVisitor<T>
            implements IVisitor<T> {
      T visit(Element e) {
         T rv = null;
         for(Element child : e.getChildren()) {
            rv = child.accept(this);
         }
         return rv;
      }
}
```

```
interface INode {
      <T> T accept(Visitor<T> v);
}
class Element implements INode {
       <T> T accept(Visitor<T> v) {
            return visitor.visit(this);
      }
}
```