Crystal Overview, Continued

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

Crystal Variable Bindings

- Variable Uses
  - Represented as ASTNodes
    - Like every other expression
    - Technically, a SimpleName
- Binding
  - A single canonical object representing the variable declaration
    - Similarly, have bindings for classes, methods…
  - Get using:
    - Name.resolveBinding()
    - VariableDeclaration.resolveBinding()
  - Eclipse doesn't provide a way to get from the Binding to the ASTNode variable declaration
    - Efficiency choice
    - Crystal shortcut
      - ASTNode Utilities.getASTNode(IBinding b)

Demo

- Installing Crystal
- Run Assignment 0
- Look at Assignment 0 code
- Look at Visitor
- Other sample analysis

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);
    }
}
```

Visitor Example

```
class StringConcatLoopVisitor extends ASTVisitor {
	int loopLevel = 0;
	public boolean visit(ForStatement node) {
		loopLevel++;
		return true;
	}
	public void endVisit(ForStatement node) {
		loopLevel--;
	}

	public boolean visit(InfixExpression node) {
		ITypeBinding type = node.getLeftOperand().resolveTypeBinding();
		if (loopLevel > 0
				&& node.getOperator() == InfixExpression.Operator.PLUS
				&& type.getName().equals("String"))
				Crystal.getInstance().reportUserProblem("Performance issue:
	String concatenation in loop (use StringBuffer instead)", node,
	StringConcatLoopAnalysis.this);
		return true;
	}
}
```


Program Semantics

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich

Why Semantics?

- Semantics describe formally what a program means
  - Typically, how the program executes
- Framework for analysis
  - Precise definitions
  - Proofs of correctness
- Semantics in practice
  - Difficult to define for full languages
    - But see Standard ML!
  - Very useful for thinking about how analysis applies to the "core" of a language
    - Extension to full language is assumed to be easy
      - **Sometimes true, sometimes not!**

Forms of Program Semantics

- Big-Step Reduction Semantics
  - Shows result of program
  - Depends on environment η : **Var → Value**
  - Form: $\eta \vdash a \downarrow v$
    - Read: In environment η, expression *a* reduces to value *v*
- Small-Big-Step Reduction Semantics
  - Shows step-by-step execution of program
  - Form: $(\eta, e) \mapsto (\eta', e')$
    - Read: In environment η, expression *e* steps to expression *e*′ and produces a new environment η'
- Denotational Semantics
  - Not covered in this course

WHILE Statement Big-Step Semantics

- We use big-step to show expression evaluation
- Inference rule format
  - Premises above the line
  - Conclusion below the line
  - Read, "If premises, then conclusion"
- Example: operators
  - If expression *a* evaluates to value *v*
  - And expression *a′* evaluates to value *v′*
  - Then the whole expression evaluates to *v* **op** *v′*, where **op** is the mathematical operator corresponding to *op*

$$\frac{\eta \vdash a \downarrow v \qquad \eta \vdash a' \downarrow v'}{\eta \vdash a \ op \ a' \downarrow v \ \mathbf{op} \ v'}$$

WHILE Expression Big-Step Semantics

- Values reduce to themselves
  - *n*, *true*, *false*
- Variables *x* reduce to value from environment
  - $\eta(x)$
- Operators reduce according to mathematical operators
  - $+, -, *, /, not, and, or, <, \leq, =, \ldots$
  - **boldface** indicates math
  - *italics* indicates program text

$$\eta \vdash n \downarrow n$$

$$\frac{\eta(x) = v}{\eta \vdash x \downarrow v}$$

$$\frac{\eta \vdash a \downarrow v \qquad \eta \vdash a' \downarrow v'}{\eta \vdash a \ op \ a' \downarrow v \ \mathbf{op} \ v'}$$

$$\eta \vdash true \downarrow \mathbf{true}$$

$$\eta \vdash false \downarrow \mathbf{false}$$

$$\frac{\eta \vdash b \downarrow v}{\eta \vdash not \ b \downarrow \mathbf{not} \ v}$$

$$\frac{\eta \vdash b \downarrow v \qquad \eta \vdash b' \downarrow v'}{\eta \vdash b \ op \ b' \downarrow v \ \mathbf{op} \ v'}$$

Applying Semantic Rules

- A tree of inference rules forms a derivation
  - Rules at top are axioms; they have no premises
- Example:
  - $\eta = [x \mapsto 3, y \mapsto 5]$
  - $b = x + 3 > y$

$$\dfrac{\dfrac{\dfrac{\eta(x) = \mathbf{3}}{\eta \vdash x \downarrow \mathbf{3}} \quad \dfrac{}{\eta \vdash 3 \downarrow \mathbf{3}}}{\eta \vdash x + 3 \downarrow \mathbf{6}} \quad \dfrac{\eta(y) = \mathbf{5}}{\eta \vdash y \downarrow \mathbf{5}}}{\eta \vdash x + 3 > y \downarrow \mathbf{true}}$$

Applying Semantic Rules

- Example:
  - $\eta = [x \mapsto 3, y \mapsto 5]$
  - $b = x > y$ *and true*

$$\eta \vdash n \downarrow n$$

$$\frac{\eta(x) = v}{\eta \vdash x \downarrow v}$$

$$\frac{\eta \vdash a \downarrow v \quad \eta \vdash a' \downarrow v'}{\eta \vdash a \ op\ a' \downarrow v \ \mathbf{op}\ v'}$$

$$\eta \vdash \mathit{true} \downarrow \mathbf{true}$$

$$\eta \vdash \mathit{false} \downarrow \mathbf{false}$$

$$\frac{\eta \vdash b \downarrow v}{\eta \vdash \mathit{not}\ b \downarrow \mathbf{not}\ v}$$

$$\frac{\eta \vdash b \downarrow v \quad \eta \vdash b' \downarrow v'}{\eta \vdash b \ op\ b' \downarrow v \ \mathbf{op}\ v'}$$

WHILE Statement Small-Step Semantics

- We use small-step for statements to show loops evaluating
- Example: assignment
  - If the right-hand side reduces to value *v*
  - Then the assignment reduces to a skip, and a new environment where *x* maps to *v*

$$\frac{\eta \vdash a \downarrow v}{(\eta,\ x{:=}a) \ \mapsto\ (\eta[x{\mapsto}v],\ skip)}$$

WHILE Statement Small-Step Semantics

- sequences reduce first statement
- *skip* is skipped if first in sequence
- *if* reduces to either first or second statement, depending on *b*
- *while* reduces to the body followed by the loop if *b* is **true**, otherwise *skip*

$$\frac{\eta \vdash a \downarrow v}{(\eta,\ x{:=}a) \ \mapsto\ (\eta[x{\mapsto}v],\ skip)}$$

$$\frac{(\eta,\ S_1) \ \mapsto\ (\eta',\ S_1')}{(\eta,\ S_1; S_2) \ \mapsto\ (\eta',\ S_1'; S_2)}$$

$$\frac{}{(\eta,\ skip; S_2) \ \mapsto\ (\eta,\ S_2)}$$

$$\frac{\eta \vdash b \downarrow \mathbf{true}}{(\eta,\ if\ b\ then\ S_1\ else\ S_2) \ \mapsto\ (\eta,\ S_1)}$$

$$\frac{\eta \vdash b \downarrow \mathbf{false}}{(\eta,\ if\ b\ then\ S_1\ else\ S_2) \ \mapsto\ (\eta,\ S_2)}$$

$$\frac{\eta \vdash b \downarrow \mathbf{true}}{(\eta,\ while\ b\ do\ S) \ \mapsto\ (\eta,\ S; while\ b\ do\ S}$$

$$\frac{\eta \vdash b \downarrow \mathbf{false}}{(\eta,\ while\ b\ do\ S) \ \mapsto\ (\eta,\ skip)}$$

WHILE Execution Example

$([],\ x := 5;\ if\ x > 3\ then\ y := 1\ else\ y := 5)$

$\mapsto_1 ([x\mapsto 5],\ skip;\ if\ x > 3\ then\ y := 1\ else\ y := 5)$

$\mapsto_2 ([x\mapsto 5],\ if\ x > 3\ then\ y := 1\ else\ y := 5)$

$\mapsto_3 ([x\mapsto 5],\ y := 1)$

$\mapsto ([x\mapsto 5, y\mapsto 1],\ skip)$

$$\cfrac{\cfrac{[] \vdash 5 \downarrow 5}{([],\ x := 5 \ \mapsto\ ([x\mapsto 5],\ skip))}}{([],\ x := 5;\ if \ldots) \ \mapsto\ ([x\mapsto 5],\ skip;\ if \ldots)}\ 1$$

$$\cfrac{}{([x\mapsto 5],\ skip;\ if \ldots) \ \mapsto\ ([x\mapsto 5],\ if \ldots)}\ 2$$

$$\cfrac{\cfrac{\cfrac{[x\mapsto 5](x) = 5}{[x\mapsto 5] \vdash x \downarrow 5} \quad \cfrac{}{[x\mapsto 5] \vdash 3 \downarrow 3}}{[x\mapsto 5] \vdash x > 3 \downarrow \mathbf{true}}}{([x\mapsto 5],\ if\ x > 3\ then\ y := 1\ else\ y := 5) \ \mapsto\ ([x\mapsto 5],\ y := 1)}\ 3$$

WHILE Execution Example

([], y := 1; x := 2; while x > 1 do y := y*x; x := x-1)

↦( )

↦( )

↦( )

↦( )

↦( )

↦( )

Proofs using WHILE Semantics

Theorem: ([y↦1, x↦n], while x > 1 do y := y*x; x := x-1)
↦* ([y↦n!, x↦1], skip)

Proof: By induction on n.

Base case (n=1):
([y↦1, x↦1], while x > 1 do y := y*x; x := x-1)
↦ ([y↦1, x↦1], skip)

Inductive case (assume induction hypothesis for n-1):
([y↦1, x↦n], while x > 1 do y := y*x; x := x-1)
↦ ([y↦1, x↦n], y := y*x; x := x-1; while x > 1 do y := y*x; x := x-1)
↦ ([y↦n, x↦n], x := x-1; while x > 1 do y := y*x; x := x-1)
↦ ([y↦n, x↦n-1], while x > 1 do y := y*x; x := x-1)
// Oops, doesn't match induction hypothesis!

Proofs using WHILE Semantics

(minor corrections from class to incorporate strengthened induction hypothesis)

Theorem: ([y↦1, x↦n], while x > 1 do y := y*x; x := x-1)
↦* ([y↦n!, x↦1], skip)

Proof: By induction on n. Strengthened induction hypothesis:
([y↦**m**, x↦n], while x > 1 do y := y*x; x := x-1)
↦* ([y↦**m*n!**, x↦1], skip)

Base case (n=1):
([y↦m, x↦1], while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*1!, x↦1], skip)

Inductive case (assume induction hypothesis for n-1):
([y↦m, x↦n], while x > 1 do y := y*x; x := x-1)
↦ ([y↦m, x↦n], y := y*x; x := x-1; while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*n, x↦n], x := x-1; while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*n, x↦n-1], while x > 1 do y := y*x; x := x-1)
↦ ([y↦m*n*(n-1)!, x↦1], skip) *// using induction hypothesis*
↦ ([y↦m*n!, x↦1], skip) *// arithmetic simplification* □

Questions?

Administrivia

- Office Hours
  - Jonathan Aldrich
    - After class
    - Thursday 2-3pm
  - Thomas LaToza
    - Monday 11am-12pm
  - Gabriel Zenarosa
    - Tuesday 5-6pm
- Assignment 0 due Tuesday, midnight
- Assignment 1 due Thursday 5pm