Implementing Abstract Dependent Classes

Matthias Krebs

Sep 04, 2019

Overview

Object Oriented Programming

```
abstract class Space {
    abstract Point getOrigin();
}
class Space2D extends Space {
    Point2D getOrigin() { return new Point2D(); }
}
class Space3D extends Space {
    Point3D getOrigin() { return new Point3D(); }
}

abstract class Point {}
class Point2D extends Point { ... }
class Point3D extends Point { ... }
```

Dependent Classes

```
abstract class Space {
  abstract Point(s: this) getOrigin();
}
class 2DSpace extends Space {
  Point(s: this) getOrigin() { ... }
}
class 3DSpace extends Space { ... }

abstract class Point(Space s) {
  abstract Point(s: s) add(Vector(s: s) v);
}
abstract class Vector(Space s) { ... }

abstract class Point(2DSpace s) { ... }
abstract class Point(3DSpace s) { ... }
```

DC_C Calculus: Structure

Constraint System

Operational Semantics   Type Relation
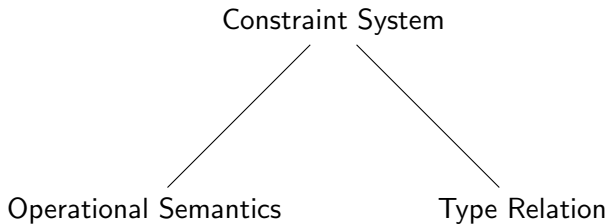
DC_C Calculus: General Syntax

$$p, q \in Path ::= x \mid p.f$$
$$a, b, c \in Constr ::= p \equiv q$$
$$\mid p :: C$$
$$\mid p.\mathbf{cls} \equiv C$$
$$t \in Type ::= [x.\ \overline{a}]$$

DC_C Calculus: Expressions & Programs

$$P \in Program ::= \overline{D}$$

$$e \in Expr ::= x \qquad\qquad D \in Decl ::= C(x.\, \overline{a})$$
$$| \; e.f \qquad\qquad\qquad | \; \forall x.\, \overline{a} \Rightarrow a$$
$$| \; \textbf{new } C(\overline{f} \equiv \overline{e}) \qquad\qquad | \; m(x.\, \overline{a}) : t$$
$$| \; m(e) \qquad\qquad\qquad | \; m(x.\, \overline{a}) : t := e$$

DC_C Calculus: Natural Numbers Program

$\text{Zero}(x.\ \epsilon)$

$\forall x.\ x :: \text{Zero} \Rightarrow x :: \text{Nat}$

$\text{Succ}(x.\ x.p :: \text{Nat})$

$\forall x.\ x :: \text{Succ}, x.p :: \text{Nat} \Rightarrow x :: \text{Nat}$

$\text{prev}(x.\ x :: \text{Nat}) : [y.\ y :: \text{Nat}]$

$\text{prev}(x.\ x :: \text{Zero}) : [y.\ y :: \text{Nat}] := \textbf{new}\ \text{Zero}()$

$\text{prev}(x.\ x :: \text{Succ}, x.p :: \text{Nat}) : [y.\ y :: \text{Nat}] := x.p$

Constraint System

$$\frac{}{a \vdash a}\ \text{(C-Ident)} \qquad\qquad \frac{}{\epsilon \vdash p \equiv p}\ \text{(C-Refl)}$$

$$\frac{\overline{a} \vdash p.\mathbf{cls} \equiv C}{\overline{a} \vdash p :: C}\ \text{(C-Class)} \qquad \frac{\overline{a} \vdash c \qquad \overline{a'}, c \vdash b}{\overline{a}, \overline{a'} \vdash b}\ \text{(C-Cut)}$$

$$\frac{\overline{a} \vdash a_{\{\!\{x \mapsto p\}\!\}} \qquad \overline{a} \vdash p' \equiv p}{\overline{a} \vdash a_{\{\!\{x \mapsto p'\}\!\}}}\ \text{(C-Subst)}$$

$$\frac{(\forall x.\ \overline{a} \Rightarrow a) \in P \qquad \overline{b} \vdash \overline{a}_{\{\!\{x \mapsto p\}\!\}}}{\overline{b} \vdash a_{\{\!\{x \mapsto p\}\!\}}}\ \text{(C-Prog)}$$

$$\overline{a} \vdash \overline{b} := \bigwedge_{b \in \overline{b}} \overline{a} \vdash b$$

Constraint System: Application

$$\frac{\overline{a} \vdash a_{\{\!\{x \mapsto p\}\!\}} \quad \overline{a} \vdash p' \equiv p}{\overline{a} \vdash a_{\{\!\{x \mapsto p'\}\!\}}} \ (\text{C-Subst})$$

$$\frac{...}{x :: \texttt{Zero}, y \equiv x \vdash y :: \texttt{Zero}}$$

Constraint System: Application

$$\frac{\overline{a} \vdash a_{\{\!\{x \mapsto p\}\!\}} \qquad \overline{a} \vdash p' \equiv p}{\overline{a} \vdash a_{\{\!\{x \mapsto p'\}\!\}}} \text{ (C-Subst)}$$

$$\frac{\dfrac{\overline{ctx} \vdash x :: \texttt{Zero}}{\overline{ctx} \vdash y :: \texttt{Zero}_{\{\!\{y \mapsto x\}\!\}}} \qquad \overline{ctx} \vdash y \equiv x}{x :: \texttt{Zero}, y \equiv x \vdash y :: \texttt{Zero}_{\{\!\{y \mapsto y\}\!\}}} \text{ C-Subst}$$

First-order Model

$$x, f, m, C \Rightarrow String$$
$$\overline{a} \vdash b \Rightarrow List[Constraint] \vdash Constraint$$
$$a, \overline{a} \Rightarrow a :: \overline{a}$$
$$a, b \Rightarrow [a, b]$$

First-order Model

$$\frac{}{a \vdash a} \text{ (C-Ident)}$$

$$\forall c : Constraint. \; [c] \vdash c$$

$$\frac{\overline{a} \vdash p.\textbf{cls} \equiv C}{\overline{a} \vdash p :: C} \text{ (C-Class)}$$

$$\forall \overline{a} : List[Constraint], p : Path, C : String. \\ \overline{a} \vdash p.\textbf{cls} \equiv C \rightarrow \overline{a} \vdash p :: C$$

First-order Model: C-Subst

$$\frac{\overline{a} \vdash a_{\{\!\{x \mapsto p\}\!\}} \qquad \overline{a} \vdash p' \equiv p}{\overline{a} \vdash a_{\{\!\{x \mapsto p'\}\!\}}} \text{ (C-Subst)}$$

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \land \overline{a} \vdash p_2 \equiv p_1$
$\qquad \land\ a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \land a_{\{\!\{x \mapsto p_2\}\!\}} = a_2$
$\quad \to \overline{a} \vdash a_2$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \text{Zero}, y \equiv x]) \vdash y :: \text{Zero}$

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \wedge \overline{a} \vdash p_2 \equiv p_1$
$\quad\quad \wedge\, a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = a_2$
$\quad\; \rightarrow \overline{a} \vdash a_2$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \mathtt{Zero}, y \equiv x]) \vdash y :: \mathtt{Zero}$

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \wedge \overline{a} \vdash p_2 \equiv p_1$
$\qquad \wedge\, a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = a_2$
$\quad \rightarrow \overline{a} \vdash a_2$

$\overline{a} := \overline{ctx}, a_2 := y :: \mathtt{Zero}$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \mathtt{Zero}, y \equiv x]) \vdash y :: \mathtt{Zero}$

$$\forall x : String, p_1, p_2 : Path, a, a_1 : Constraint.$$
$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash p_2 \equiv p_1$$
$$\wedge\; a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = y :: \mathtt{Zero}$$
$$\rightarrow \overline{ctx} \vdash y :: \mathtt{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \mathtt{Zero}$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \mathtt{Zero}, y \equiv x]) \vdash y :: \mathtt{Zero}$

$$\forall x : String, p_1, p_2 : Path, a, a_1 : Constraint.$$
$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash p_2 \equiv p_1$$
$$\wedge\ a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = y :: \mathtt{Zero}$$
$$\rightarrow \overline{ctx} \vdash y :: \mathtt{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \mathtt{Zero}$
$x := \texttt{"y"}, p_2 = y, p_1 = x$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \text{Zero}, y \equiv x]) \vdash y :: \text{Zero}$

$$\forall a, a_1 : \textit{Constraint}.$$
$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$
$$\wedge\ a_{\{"y" \mapsto x\}} = a_1 \wedge a_{\{"y" \mapsto y\}} = y :: \text{Zero}$$
$$\rightarrow \overline{ctx} \vdash y :: \text{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \text{Zero}$
$x := \text{"y"}, p_2 = y, p_1 = x$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \mathtt{Zero}, y \equiv x]) \vdash y :: \mathtt{Zero}$

$$\forall a, a_1 : \textit{Constraint}.$$
$$\overline{ctx} \vdash a_1 \land \overline{ctx} \vdash y \equiv x$$
$$\land\ a_{\{\texttt{"y"}\mapsto x\}} = a_1 \land a_{\{\texttt{"y"}\mapsto y\}} = y :: \mathtt{Zero}$$
$$\rightarrow \overline{ctx} \vdash y :: \mathtt{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \mathtt{Zero}$
$x := \texttt{"y"}, p_2 = y, p_1 = x$
$a := y :: \mathtt{Zero}$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \texttt{Zero}, y \equiv x]) \vdash y :: \texttt{Zero}$

$$\forall a_1 : \textit{Constraint}.$$
$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$
$$\wedge \; y :: \texttt{Zero}_{\{"y" \mapsto x\}} = a_1$$
$$\rightarrow \overline{ctx} \vdash y :: \texttt{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \texttt{Zero}$

$x := "y", p_2 = y, p_1 = x$

$a := y :: \texttt{Zero}$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \text{Zero}, y \equiv x]) \vdash y :: \text{Zero}$

$$\forall a_1 : Constraint.$$
$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$
$$\wedge \, y :: \text{Zero}_{\{"y" \mapsto x\}} = a_1$$
$$\rightarrow \overline{ctx} \vdash y :: \text{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \text{Zero}$

$x := "y", p_2 = y, p_1 = x$

$a := y :: \text{Zero}$

$a_1 := x :: \text{Zero}$

First-order Model: Application

To show: $(\overline{ctx} := [x :: \text{Zero}, y \equiv x]) \vdash y :: \text{Zero}$

$$\overline{ctx} \vdash x :: \text{Zero} \wedge \overline{ctx} \vdash y \equiv x$$
$$\rightarrow \overline{ctx} \vdash y :: \text{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \text{Zero}$

$x := \text{"y"}, p_2 = y, p_1 = x$

$a := y :: \text{Zero}$

$a_1 := x :: \text{Zero}$

Application Comparison

$$\dfrac{\dfrac{\overline{ctx} \vdash x :: \texttt{Zero}}{\overline{ctx} \vdash y :: \texttt{Zero}_{\{y \mapsto x\}}} \quad \overline{ctx} \vdash y \equiv x}{x :: \texttt{Zero}, y \equiv x \vdash y :: \texttt{Zero}_{\{y \mapsto y\}}} \text{ C-Subst}$$

$$\overline{ctx} \vdash x :: \texttt{Zero} \wedge \overline{ctx} \vdash y \equiv x \qquad \text{(C-Subst)}$$
$$\rightarrow \overline{ctx} \vdash y :: \texttt{Zero}$$

Analyze C-Subst

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \wedge \overline{a} \vdash p_2 \equiv p_1$
$\qquad \wedge\ a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = a_2$
$\quad \rightarrow \overline{a} \vdash a_2$

Analyze C-Subst

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
 $\quad \overline{a} \vdash a_1 \wedge \overline{a} \vdash p_2 \equiv p_1$
 $\qquad \wedge\ a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = a_2$
 $\quad \rightarrow \overline{a} \vdash a_2$

$a_2 := <input>\ ,\ a_{\{\!\{x \mapsto p_2\}\!\}} := a_2\ ,\ a_1 := a_{\{\!\{x \mapsto p_1\}\!\}}$

Analyze C-Subst

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \wedge \overline{a} \vdash p_2 \equiv p_1$
$\qquad \wedge\ a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$
$\quad \rightarrow \overline{a} \vdash a_2$

$a_2 := <input>$, $a_{\{x \mapsto p_2\}} := a_2$, $a_1 := a_{\{x \mapsto p_1\}}$
Instantiation Pattern: $a_2 \rightarrow a \rightarrow a_1$

Generalization

Substitution: $a_{\{x \mapsto p\}}$
Generalization: $a_{\{p \mapsto x\}}$

$$(a_{\{x \mapsto p\}})_{\{p \mapsto x\}} = a$$

Generalization

Substitution: $a_{\{x \mapsto p\}}$
Generalization: $a_{\{p \mapsto x\}}$

$$(a_{\{x \mapsto p\}})_{\{p \mapsto x\}} = a$$

$$x.g :: C_{\{x \mapsto y.f\}} = y.f.g :: C$$
$$y.f.g :: C_{\{y.f \mapsto x\}} = x.g :: C$$
$$y.f.g :: C_{\{f.g \mapsto x\}} = y.x :: C \qquad \text{(not possible)}$$

C-Subst: Algorithmic Instantiations

Instantiation Pattern: $a_2 \rightarrow a \rightarrow a_1$

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \wedge \overline{a} \vdash p_2 \equiv p_1$
$\quad\quad \wedge\, a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = a_2$
$\quad \rightarrow \overline{a} \vdash a_2$

$$a_2 := <input>$$
$$a_{\{\!\{x \mapsto p_2\}\!\}} := a_2$$
$$a_1 := a_{\{\!\{x \mapsto p_1\}\!\}}$$

C-Subst: Algorithmic Instantiations

Instantiation Pattern: $a_2 \to a \to a_1$

$\forall \overline{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$
$\quad \overline{a} \vdash a_1 \land \overline{a} \vdash p_2 \equiv p_1$
$\quad\quad \land a_{\{x \mapsto p_1\}} = a_1 \land a = a_{2\{p_2 \mapsto x\}}$
$\quad \to \overline{a} \vdash a_2$

$$a_2 := <input>$$
$$a := a_{2\{p_2 \mapsto x\}}$$
$$a_1 := a_{\{x \mapsto p_1\}}$$

C-Subst: Algorithmic Instantiations

Instantiation Pattern: $a_2 \rightarrow a \rightarrow a_1$

$\forall \overline{a} : List[Constraint], a_2 : Constraint, x : String, p_1, p_2 : Path.$
   $\textbf{let } a := a_{2\{p_2 \mapsto x\}} \textbf{ in}$
     $\textbf{let } a_1 := a_{\{x \mapsto p_1\}}$
     $\textbf{in } \overline{a} \vdash p_2 \equiv p_1 \wedge \overline{a} \vdash a_1$
 $\rightarrow \overline{a} \vdash a_2$

$$a_2 := <input>$$
$$a := a_{2\{p_2 \mapsto x\}}$$
$$a_1 := a_{\{x \mapsto p_1\}}$$

Operational Semantics

$$S = \{\langle \overline{a}; e \rangle \mid \langle \overline{a}; e \rangle \in MImpl(m, x) \wedge HC(h) \vdash \overline{a}\} \qquad \langle \overline{a}; e \rangle \in S$$

$$\frac{\forall \langle \overline{a'}; e' \rangle \in S. \ (e' \neq e) \longrightarrow (\overline{a'} \vdash \overline{a}) \wedge \neg(\overline{a} \vdash \overline{a'})}{\langle h; m(x) \rangle \rightarrow \langle h; e \rangle} \ \text{R-Call}$$

$$o ::= \langle C; \overline{f} \equiv \overline{x} \rangle \qquad\qquad \text{(objects)}$$
$$h ::= \overline{x} \mapsto \overline{o} \ \ (x_i \text{ distinct}) \qquad \text{(heaps)}$$

$$OC(x, o) = (x.\textbf{cls} \equiv C, x.\overline{f} \equiv \overline{x})$$
$$HC(h) = \bigcup_i OC(x_i, o_i)$$

Operational Semantics: Implementation

```
def interp(heap: Heap, expr: Expression)
          : (Heap, Expression) = expr match {
  case ...
  case MethodCall(m, x@Id(_)) =>    // R-Call
    // Applicable methods
    val S = mImplSubst(m, x).filter {
      case (as, _) => entails(HC(heap), as) }

    if (S.isEmpty) // m not in program
      return (heap, expr)

    var (a, e) = S.head // Most specific method
    S.foreach {
      ...
    }
    interp(heap, e)
  case ...
}
```

Operational Semantics: Application

$h := [x \rightarrow (\text{Zero}, \textit{nil}),$ $e := \text{prev}(y)$

      $y \rightarrow (\text{Succ}, [(p, x)])]$ $HC(h) := [x.\textbf{cls} \equiv \text{Zero}, y.\textbf{cls} \equiv \text{Succ}, y.p \equiv x]$

```
    case MethodCall(m, x@Id(_)) =>    // R-Call
      val S = mImplSubst(m, x).filter {
        case (as, _) => entails(HC(heap), as) }
```

Operational Semantics: Application

$$h := [x \to (\text{Zero}, \textit{nil}), \qquad\qquad e := \text{prev}(y)$$
$$\quad\; y \to (\text{Succ}, [(p, x)])] \quad HC(h) := [x.\mathbf{cls} \equiv \text{Zero}, y.\mathbf{cls} \equiv \text{Succ}, y.p \equiv x]$$

```
case MethodCall(m, x@Id(_)) =>    // R-Call
  val S = mImplSubst(m, x).filter {
    case (as, _) => entails(HC(heap), as) }
```

$$\text{prev}(x.\; x :: \text{Zero}) : [y.\; y :: \text{Nat}] := \textbf{new } \text{Zero}()$$
$$\text{prev}(x.\; x :: \text{Succ}, x.p :: \text{Nat}) : [y.\; y :: \text{Nat}] := x.p$$

Operational Semantics: Application

$$h := [x \rightarrow (\text{Zero}, \textit{nil}),$$
$$\qquad y \rightarrow (\text{Succ}, [(p, x)])] \quad HC(h) := [x.\textbf{cls} \equiv \text{Zero}, y.\textbf{cls} \equiv \text{Succ}, y.p \equiv x]$$

$$e := \text{prev}(y)$$

```
case MethodCall(m, x@Id(_)) =>    // R-Call
  val S = mImplSubst(m, x).filter {
    case (as, _) => entails(HC(heap), as) }
```

$$\text{prev}(x.\ x :: \text{Zero}) : [y.\ y :: \text{Nat}] := \textbf{new } \text{Zero}()$$
$$\text{prev}(x.\ x :: \text{Succ}, x.p :: \text{Nat}) : [y.\ y :: \text{Nat}] := x.p$$

$$HC(h) \vdash y :: \text{Zero}$$

Operational Semantics: Application

$$h := [x \to (\text{Zero}, \mathit{nil}), \qquad\qquad e := \text{prev}(y)$$
$$\qquad y \to (\text{Succ}, [(p, x)])] \quad HC(h) := [x.\textbf{cls} \equiv \text{Zero}, y.\textbf{cls} \equiv \text{Succ}, y.p \equiv x]$$

```
case MethodCall(m, x@Id(_)) =>    // R–Call
  val S = mImplSubst(m, x).filter {
    case (as, _) => entails(HC(heap), as) }
```

$$\text{prev}(x.\ x :: \text{Zero}) : [y.\ y :: \text{Nat}] := \textbf{new Zero}()$$
$$\text{prev}(x.\ x :: \text{Succ}, x.p :: \text{Nat}) : [y.\ y :: \text{Nat}] := x.p$$

$$HC(h) \vdash y :: \text{Zero}$$
$$HC(h) \vdash y :: \text{Succ}$$
$$HC(h) \vdash y.p :: \text{Nat}$$

Type Assignment

$$\frac{\begin{array}{ccc} \forall i.\ \overline{c} \vdash e_i : [x_i.\ \overline{a_i}] & & C(x.\ \overline{b'}) \in P \\ \overline{b} = (x.\textbf{cls} \equiv C), \bigcup_i \overline{a_i}_{\{\!\{x_i \mapsto x.f_i\}\!\}} & & \overline{c}, \overline{b} \vdash \overline{b'} \end{array}}{\overline{c} \vdash \textbf{new}\ C(\overline{f} \equiv \overline{e}) : [x.\ \overline{b}]}\ \text{T-New}$$

$$\frac{\overline{c} \vdash e : [x.\ \overline{a'}] \qquad \overline{c}, \overline{a'} \vdash \overline{a}}{\overline{c} \vdash e : [x.\ \overline{a}]}\ \text{T-Sub}$$

Type Assignment

$$\forall i.\ \overline{c} \vdash e_i : [x_i.\ \overline{a_i}] \qquad\qquad C(x.\ \overline{b'}) \in P$$
$$\frac{\overline{b} = (x.\textbf{cls} \equiv C), \bigcup_i \overline{a_i}_{\{x_i \mapsto x.f_i\}} \qquad \overline{c}, \overline{b} \vdash \overline{b'}}{\overline{c} \vdash \textbf{new}\ C(\overline{f} \equiv \overline{e}) : [x.\ \overline{b}]}\ \text{T-New}$$

$$\frac{\overline{c} \vdash e : [x.\ \overline{a'}] \qquad \overline{c}, \overline{a'} \vdash \overline{a}}{\overline{c} \vdash e : [x.\ \overline{a}]}\ \text{T-Sub}$$

Implementation:

typeassignment(context : List [Constraint], exp : Expression) : List [Type]

Type Assignment: Example

$$\frac{\forall i. \ \overline{c} \vdash e_i : [x_i. \ \overline{a_i}] \qquad C(x. \ \overline{b'}) \in P}{\overline{b} = (x.\textbf{cls} \equiv C), \bigcup_i \overline{a_i}_{\{x_i \mapsto x.f_i\}} \qquad \overline{c}, \overline{b} \vdash \overline{b'}}{\overline{c} \vdash \textbf{new} \ C(\overline{f} \equiv \overline{e}) : [x. \ \overline{b}]} \ \text{T-New}$$

Types of **new** Zero()

Type Assignment: Example

$$\frac{\forall i. \ \overline{c} \vdash e_i : [x_i. \ \overline{a_i}] \qquad C(x. \ \overline{b'}) \in P}{\overline{b} = (x.\mathbf{cls} \equiv C), \bigcup_i \overline{a_i}_{\{\!\{x_i \mapsto x.f_i\}\!\}} \qquad \overline{c}, \overline{b} \vdash \overline{b'}}{\overline{c} \vdash \mathbf{new} \ C(\overline{f} \equiv \overline{e}) : [x. \ \overline{b}]} \ \text{T-New}$$

Types of **new** Zero()

$$[x. \ x.\mathbf{cls} \equiv \mathtt{Zero}]$$

Type Assignment: Example

$$\frac{\forall i.\ \overline{c} \vdash e_i : [x_i.\ \overline{a_i}] \qquad C(x.\ \overline{b'}) \in P}{\overline{b} = (x.\textbf{cls} \equiv C), \bigcup_i \overline{a_i}_{\{x_i \mapsto x.f_i\}} \qquad \overline{c}, \overline{b} \vdash \overline{b'}}{\overline{c} \vdash \textbf{new}\ C(\overline{f} \equiv \overline{e}) : [x.\ \overline{b}]}\ \text{T-New}$$

Types of **new** Zero()

$$[x.\ x.\textbf{cls} \equiv \texttt{Zero}]$$
$$[x.\ x :: \texttt{Zero}]$$
$$[x.\ x :: \texttt{Nat}]$$

Well-formedness

$$\frac{FV(\overline{a}) = \{x\} \qquad FV(\overline{b}) = \{x, y\} \qquad \overline{a} \vdash e : [y.\ \overline{b}]}{\mathsf{wf}\ (m(x.\ \overline{a}) : [y.\ \overline{b}] := e)}\ \text{WF-MI}$$

Well-formedness

$$\frac{FV(\overline{a}) = \{x\} \qquad FV(\overline{b}) = \{x, y\} \qquad \overline{a} \vdash e : [y.\ \overline{b}]}{\text{wf } (m(x.\ \overline{a}) : [y.\ \overline{b}] := e)} \text{ WF-MI}$$

```
def wf(D: Declaration): Boolean = D match {
  case ...
  case MethodImplementation(_, x, a, Type(y, b), e) =>
    FV(a) == List(x) && FV(b).nonEmpty &&
      FV(b).forall(v => v == x || v == y) &&
    typeassignment(a, e).exists {
      case Type(z, c) =>
        c.size == b.size &&
        substitute(z, y, c).forall(b.contains(_))
    }
}
```

Well-formedness: Example

$$\frac{FV(\overline{a}) = \{x\} \qquad FV(\overline{b}) = \{x, y\} \qquad \overline{a} \vdash e : [y.\ \overline{b}]}{\text{wf } (m(x.\ \overline{a}) : [y.\ \overline{b}] := e)} \text{ WF-MI}$$

$$\text{prev}(x.\ x :: \text{Zero}) : [y.\ y :: \text{Nat}] := \textbf{new Zero}()$$

$$\begin{aligned}
type(\textbf{new Zero}()) := [&[x.\ x.\textbf{cls} \equiv \text{Zero}], \\
&[x.\ x :: \text{Zero}], \\
&[x.\ x :: \text{Nat}]]
\end{aligned}$$

Context Enrichment

$$\overline{a} \vdash a$$

Context Enrichment

$$\overline{a} \vdash a$$
$$\overline{b} + \overline{a} \vdash a$$

Context Enrichment

$$\overline{a} \vdash a$$
$$\overline{b} + \overline{a} \vdash a$$

$$[x :: C] \vdash y :: C$$
$$[y \equiv x, x :: C] \vdash y :: C \qquad \text{(thin air)}$$

Context Enrichment

$$\overline{a} \vdash a$$
$$\overline{b} + \overline{a} \vdash a$$

$$[x :: C] \vdash y :: C$$
$$[y \equiv x, x :: C] \vdash y :: C \qquad \text{(thin air)}$$

$$[x :: \texttt{Succ}, x.p :: \texttt{Nat}] \vdash c$$
$$[x :: \texttt{Zero}, x :: \texttt{Succ}, x.p :: \texttt{Nat}] \vdash c \qquad \text{(contradiction)}$$

Information Transfer: Compile Time to Runtime

$$e : [x.\ \overline{a}]$$

Information Transfer: Compile Time to Runtime

$$e : [x.\ \overline{a}]$$
$$\overline{b} \vdash b \qquad (\text{origin: } interp(e))$$
$$\overline{a} \mathbin{+\!\!+} \overline{b} \vdash b$$

Information Transfer: Compile Time to Runtime

$$e : [x.\ \overline{a}]$$
$$\overline{b} \vdash b \qquad (\text{origin: } interp(e))$$
$$\overline{a} + \!\!\!+\ \overline{b} \vdash b$$

$$e : [y.\ y :: \mathtt{Nat}]$$
$$e \to x$$
$$[x :: \mathtt{Zero}] \vdash x :: \mathtt{Nat}$$
$$[x :: \mathtt{Nat}, x :: \mathtt{Zero}] \vdash x :: \mathtt{Nat}$$