

Implementing Abstract Dependent Classes

Matthias Krebs

Sep 04, 2019

Overview

1. Dependent Classes
2. DC_C Calculus
3. Implementation
4. Runtime Optimization

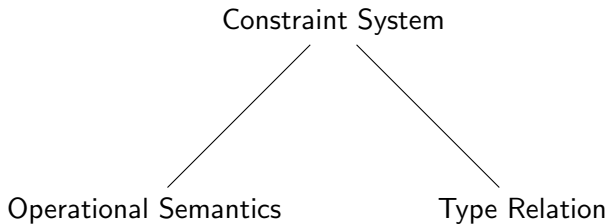
Object Oriented Programming

```
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



DC_C Calculus: General Syntax

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

DC_C Calculus: Expressions & Programs

$e \in Expr ::= x$

| $e.f$

| **new** $C(\bar{f} \equiv \bar{e})$

| $m(e)$

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

$D \in Decl ::= C(x. \bar{a})$

| $\forall x. \bar{a} \Rightarrow a$

| $m(x. \bar{a}) : t$

| $m(x. \bar{a}) : t := e$

DC_C Calculus: Natural Numbers Program

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

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

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

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

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

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

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

Constraint System

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

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

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

$$\frac{\bar{a} \vdash c \quad \bar{a}', c \vdash b}{\bar{a}, \bar{a}' \vdash b} \text{ (C-Cut)}$$

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

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

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

Constraint System: Application

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

$$\frac{\dots}{x :: \text{Zero}, y \equiv x \vdash y :: \text{Zero}}$$

Constraint System: Application

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

$$\frac{\dots}{x :: \text{Zero}, y \equiv x \vdash y :: \text{Zero}}$$

Constraint System: Application

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

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

First-order Model

$x, f, m, C \Rightarrow \text{String}$

$\bar{a} \vdash b \Rightarrow \text{List}[\text{Constraint}] \vdash \text{Constraint}$

$a, \bar{a} \Rightarrow a :: \bar{a}$

$a, b \Rightarrow [a, b]$

First-order Model

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

$$\forall c : \text{Constraint}. [c] \vdash c$$

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

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

First-order Model: C-Subst

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

$\forall \bar{a} : \text{List}[\text{Constraint}], x : \text{String}, p_1, p_2 : \text{Path}, a, a_1, a_2 : \text{Constraint}.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

First-order Model: Application

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

First-order Model: Application

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

$\forall \bar{a} : \text{List}[\text{Constraint}], x : \text{String}, p_1, p_2 : \text{Path}, a, a_1, a_2 : \text{Constraint}.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

First-order Model: Application

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

$\forall \bar{a} : \text{List}[\text{Constraint}], x : \text{String}, p_1, p_2 : \text{Path}, a, a_1, a_2 : \text{Constraint}.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

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

First-order Model: Application

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

$\forall x : \text{String}, p_1, p_2 : \text{Path}, a, a_1 : \text{Constraint}.$

$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash p_2 \equiv p_1$

$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = y :: \text{Zero}$

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

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

First-order Model: Application

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

$\forall x : \text{String}, p_1, p_2 : \text{Path}, a, a_1 : \text{Constraint}.$

$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash p_2 \equiv p_1$

$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = y :: \text{Zero}$

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

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

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

First-order Model: Application

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

$\forall a, a_1 : \text{Constraint.}$

$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$

$$\wedge a_{\{\text{"y"} \mapsto x\}} = a_1 \wedge a_{\{\text{"y"} \mapsto y\}} = y :: \text{Zero}$$

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

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

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

First-order Model: Application

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

$\forall a, a_1 : \text{Constraint.}$

$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$

$$\wedge a_{\{\text{"y"} \mapsto x\}} = a_1 \wedge a_{\{\text{"y"} \mapsto y\}} = y :: \text{Zero}$$

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

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

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

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

First-order Model: Application

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

$\forall a_1 : \text{Constraint.}$

$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$

$$\wedge y :: \text{Zero}_{\{\text{"y"} \mapsto x\}} = a_1$$

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

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

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

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

First-order Model: Application

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

$\forall a_1 : \text{Constraint.}$

$$\overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x$$

$$\wedge y :: \text{Zero}_{\{\text{"y"} \mapsto x\}} = a_1$$

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

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

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

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

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

First-order Model: Application

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

$$\begin{aligned} &\overline{ctx} \vdash x :: \text{Zero} \wedge \overline{ctx} \vdash y \equiv x \\ &\rightarrow \overline{ctx} \vdash y :: \text{Zero} \end{aligned}$$

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

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

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

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

Application Comparison

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

$$\begin{array}{l} \overline{ctx} \vdash x :: \text{Zero} \wedge \overline{ctx} \vdash y \equiv x \\ \rightarrow \overline{ctx} \vdash y :: \text{Zero} \end{array} \quad (\text{C-Subst})$$

Analyze C-Subst

$\forall \bar{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

Analyze C-Subst

$\forall \bar{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

$$a_2 := \langle input \rangle, a_{\{x \mapsto p_2\}} := a_2, a_1 := a_{\{x \mapsto p_1\}}$$

Analyze C-Subst

$\forall \bar{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

$a_2 := \langle input \rangle$, $a_{\{x \mapsto p_2\}} := a_2$, $a_1 := a_{\{x \mapsto p_1\}}$

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

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

$$x.g :: C_{\{x \mapsto y.f\}} = y.f.g :: C$$

$$y.f.g :: C_{\{y.f \mapsto x\}} = x.g :: C$$

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

C-Subst: Algorithmic Instantiations

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

$\forall \bar{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

C-Subst: Algorithmic Instantiations

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

$\forall \bar{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a_{\{x \mapsto p_2\}} = a_2$$

$$\rightarrow \bar{a} \vdash a_2$$

$$a_2 := <input>$$

$$a := a_2_{\{p_2 \mapsto x\}}$$

$$a_1 := a_{\{x \mapsto p_1\}}$$

C-Subst: Algorithmic Instantiations

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

$\forall \bar{a} : List[Constraint], x : String, p_1, p_2 : Path, a, a_1, a_2 : Constraint.$

$$\bar{a} \vdash a_1 \wedge \bar{a} \vdash p_2 \equiv p_1$$

$$\wedge a_{\{x \mapsto p_1\}} = a_1 \wedge a = a_2_{\{p_2 \mapsto x\}}$$

$$\rightarrow \bar{a} \vdash a_2$$

$$a_2 := \langle input \rangle$$

$$a := a_2_{\{p_2 \mapsto x\}}$$

$$a_1 := a_{\{x \mapsto p_1\}}$$

C-Subst: Algorithmic Instantiations

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

$\forall \bar{a} : List[Constraint], a_2 : Constraint, x : String, p_1, p_2 : Path.$

let $a := a_2 \{p_2 \mapsto x\}$ **in**

let $a_1 := a \{x \mapsto p_1\}$

in $\bar{a} \vdash p_2 \equiv p_1 \wedge \bar{a} \vdash a_1$

$\rightarrow \bar{a} \vdash a_2$

$a_2 := \langle input \rangle$

$a := a_2 \{p_2 \mapsto x\}$

$a_1 := a \{x \mapsto p_1\}$

Operational Semantics

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

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

$$o ::= \langle C; \bar{f} \equiv \bar{x} \rangle \quad (\text{objects})$$

$$h ::= \bar{x} \mapsto \bar{o} \quad (x_i \text{ distinct}) \quad (\text{heaps})$$

$$OC(x, o) = (x.\mathbf{cls} \equiv C, x.\bar{f} \equiv \bar{x})$$

$$HC(h) = \bigcup_i OC(x_i, o_i)$$

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

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

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

Operational Semantics: Application

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

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

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

Operational Semantics: Application

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

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

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

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

Operational Semantics: Application

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

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

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

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

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

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

Operational Semantics: Application

$$\begin{array}{ll} h := [x \rightarrow (\text{Zero}, \text{nil}), & e := \text{prev}(y) \\ y \rightarrow (\text{Succ}, [(p, x)])] & HC(h) := [x.\mathbf{cls} \equiv \text{Zero}, y.\mathbf{cls} \equiv \text{Succ}, y.p \equiv x] \\ & e_1 := y.p \end{array}$$

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

Type Assignment

$$\frac{\begin{array}{c} \forall i. \bar{c} \vdash e_i : [x_i. \bar{a}_i] \qquad C(x. \bar{b}') \in P \\ \bar{b} = (x.\mathbf{cls} \equiv C), \bigcup_i \bar{a}_i \{ \{x_i \mapsto x.f_i\} \} \qquad \bar{c}, \bar{b} \vdash \bar{b}' \end{array}}{\bar{c} \vdash \mathbf{new} \ C(\bar{f} \equiv \bar{e}) : [x. \bar{b}]} \text{ T-New}$$
$$\frac{\bar{c} \vdash e : [x. \bar{a}'] \qquad \bar{c}, \bar{a}' \vdash \bar{a}}{\bar{c} \vdash e : [x. \bar{a}]} \text{ T-Sub}$$

Type Assignment

$$\frac{\begin{array}{c} \forall i. \bar{c} \vdash e_i : [x_i. \bar{a}_i] \qquad C(x. \bar{b}') \in P \\ \bar{b} = (x.\mathbf{cls} \equiv C), \bigcup_i \bar{a}_i \{x_i \mapsto x.f_i\} \qquad \bar{c}, \bar{b} \vdash \bar{b}' \end{array}}{\bar{c} \vdash \mathbf{new} \ C(\bar{f} \equiv \bar{e}) : [x. \bar{b}]} \text{ T-New}$$
$$\frac{\bar{c} \vdash e : [x. \bar{a}'] \qquad \bar{c}, \bar{a}' \vdash \bar{a}}{\bar{c} \vdash e : [x. \bar{a}]} \text{ T-Sub}$$

Implementation:

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

Type Assignment: Example

$$\frac{\begin{array}{l} \forall i. \bar{c} \vdash e_i : [x_i. \bar{a}_i] \qquad C(x. \bar{b}') \in P \\ \bar{b} = (x.\mathbf{cls} \equiv C), \bigcup_i \bar{a}_i \{\{x_i \mapsto x.f_i\}\} \qquad \bar{c}, \bar{b} \vdash \bar{b}' \end{array}}{\bar{c} \vdash \mathbf{new} \ C(\bar{f} \equiv \bar{e}) : [x. \bar{b}]} \text{T-New}$$

Types of **new** Zero()

Type Assignment: Example

$$\frac{\begin{array}{c} \forall i. \bar{c} \vdash e_i : [x_i. \bar{a}_i] \\ \bar{b} = (x.\mathbf{cls} \equiv C), \bigcup_i \bar{a}_i \{ \{ x_i \mapsto x.f_i \} \} \end{array} \quad \begin{array}{c} C(x. \bar{b}') \in P \\ \bar{c}, \bar{b} \vdash \bar{b}' \end{array}}{\bar{c} \vdash \mathbf{new} \ C(\bar{f} \equiv \bar{e}) : [x. \bar{b}]} \text{T-New}$$

Types of **new** Zero()

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

Type Assignment: Example

$$\frac{\begin{array}{c} \forall i. \bar{c} \vdash e_i : [x_i. \bar{a}_i] \\ \bar{b} = (x.\mathbf{cls} \equiv C), \bigcup_i \bar{a}_i \{ \{ x_i \mapsto x.f_i \} \} \end{array} \quad \begin{array}{c} C(x. \bar{b}') \in P \\ \bar{c}, \bar{b} \vdash \bar{b}' \end{array}}{\bar{c} \vdash \mathbf{new} \ C(\bar{f} \equiv \bar{e}) : [x. \bar{b}]} \text{T-New}$$

Types of **new** Zero()

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

$[x. x :: \mathbf{Zero}]$

$[x. x :: \mathbf{Nat}]$

Well-formedness

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

Well-formedness

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

```
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

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

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

$\text{type}(\mathbf{new} \text{ Zero}()) := [[x. x.\mathbf{cls} \equiv \text{Zero}],$
 $[x. x :: \text{Zero}],$
 $[x. x :: \text{Nat}]]$

Context Enrichment

$$\bar{a} \vdash a$$

Context Enrichment

$$\begin{array}{c} \bar{a} \vdash a \\ \bar{b} \dashv\vdash \bar{a} \vdash a \end{array}$$

Context Enrichment

$$\begin{array}{c} \bar{a} \vdash a \\ \bar{b} \# \bar{a} \vdash a \end{array}$$

$$\begin{array}{c} [x :: C] \vdash y :: C \\ [y \equiv x, x :: C] \vdash y :: C \quad (\text{thin air}) \end{array}$$

Context Enrichment

$$\begin{array}{c} \bar{a} \vdash a \\ \bar{b} \dashv\vdash \bar{a} \vdash a \end{array}$$

$$\begin{array}{c} [x :: C] \vdash y :: C \\ [y \equiv x, x :: C] \vdash y :: C \end{array} \quad (\text{thin air})$$

$$\begin{array}{c} [x :: \text{Succ}, x.p :: \text{Nat}] \vdash c \\ [x :: \text{Zero}, x :: \text{Succ}, x.p :: \text{Nat}] \vdash c \end{array} \quad (\text{contradiction})$$

Information Transfer: Compile Time to Runtime

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

Information Transfer: Compile Time to Runtime

$$\begin{array}{l} e : [x. \bar{a}] \\ \bar{b} \vdash b \\ \bar{a} \dashv\vdash \bar{b} \vdash b \end{array} \quad (\text{origin: } \textit{interp}(e))$$

Information Transfer: Compile Time to Runtime

$$\begin{array}{l} e : [x. \bar{a}] \\ \bar{b} \vdash b \\ \bar{a} \dashv\vdash \bar{b} \vdash b \end{array} \quad (\text{origin: } \textit{interp}(e))$$

$$\begin{array}{l} e : [y. y :: \text{Nat}] \\ e \rightarrow x \\ [x :: \text{Zero}] \vdash x :: \text{Nat} \\ [x :: \text{Nat}, x :: \text{Zero}] \vdash x :: \text{Nat} \end{array}$$