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

```java
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

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

DC_C Calculus: Expressions & Programs

$$
\begin{aligned}
e \in \mathit{Expr} ::= {} & x \\
\mid {} & e.f \\
\mid {} & \textbf{new}\ C(\overline{f} \equiv \overline{e}) \\
\mid {} & m(e)
\end{aligned}
$$

$$
\begin{aligned}
P \in \mathit{Program} ::= {} & \overline{D} \\
D \in \mathit{Decl} ::= {} & C(x.\ \overline{a}) \\
\mid {} & \forall x.\ \overline{a} \Rightarrow a \\
\mid {} & m(x.\ \overline{a}) : t \\
\mid {} & m(x.\ \overline{a}) : t := e
\end{aligned}
$$

DC_C Calculus: Natural Numbers Program

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

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

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

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

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

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

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

Constraint System

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

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

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

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

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

Constraint System: Application

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

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

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

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

First-order Model

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

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

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

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

First-order Model: C-Subst

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

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

First-order Model: Application

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

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

First-order Model: Application

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

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

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

First-order Model: Application

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

$$
\begin{aligned}
&\forall x : \mathit{String}, p_1, p_2 : \mathit{Path}, a, a_1 : \mathit{Constraint}.\\
&\quad \overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash p_2 \equiv p_1\\
&\qquad \wedge\, a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = y :: \texttt{Zero}\\
&\quad \to \overline{ctx} \vdash y :: \texttt{Zero}
\end{aligned}
$$

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

First-order Model: Application

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

$$
\begin{aligned}
&\forall x : String, p_1, p_2 : Path, a, a_1 : Constraint. \\
&\quad \overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash p_2 \equiv p_1 \\
&\qquad \wedge a_{\{\!\{x \mapsto p_1\}\!\}} = a_1 \wedge a_{\{\!\{x \mapsto p_2\}\!\}} = y :: \texttt{Zero} \\
&\quad \to \overline{ctx} \vdash y :: \texttt{Zero}
\end{aligned}
$$

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

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

First-order Model: Application

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

$$
\begin{aligned}
&\forall a, a_1 : \mathit{Constraint}. \\
&\quad \overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x \\
&\qquad \wedge\, a_{\{\!\{\texttt{"y"} \mapsto x\}\!\}} = a_1 \wedge a_{\{\!\{\texttt{"y"} \mapsto y\}\!\}} = y :: \texttt{Zero} \\
&\quad \to \overline{ctx} \vdash y :: \texttt{Zero}
\end{aligned}
$$

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

First-order Model: Application

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

$$
\begin{aligned}
&\forall a, a_1 : \mathit{Constraint}.\\
&\quad \overline{ctx} \vdash a_1 \land \overline{ctx} \vdash y \equiv x\\
&\qquad \land a_{\{\!\{\texttt{"y"} \mapsto x\}\!\}} = a_1 \land a_{\{\!\{\texttt{"y"} \mapsto y\}\!\}} = y :: \texttt{Zero}\\
&\quad \to \overline{ctx} \vdash y :: \texttt{Zero}
\end{aligned}
$$

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

First-order Model: Application

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

$$\begin{aligned}
&\forall a_1 : \mathit{Constraint}.\\
&\quad \overline{ctx} \vdash a_1 \wedge \overline{ctx} \vdash y \equiv x\\
&\qquad \wedge\, y :: \texttt{Zero}_{\{\!\{\texttt{"y"} \mapsto x\}\!\}} = a_1\\
&\quad \to \overline{ctx} \vdash y :: \texttt{Zero}
\end{aligned}$$

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

First-order Model: Application

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

$$
\begin{aligned}
&\forall a_1 : \mathit{Constraint}.\\
&\quad \overline{ctx} \vdash a_1 \land \overline{ctx} \vdash y \equiv x\\
&\qquad \land\, y :: \texttt{Zero}_{\{\!\{\texttt{"y"} \mapsto x\}\!\}} = a_1\\
&\quad \to \overline{ctx} \vdash y :: \texttt{Zero}
\end{aligned}
$$

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

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

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

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

First-order Model: Application

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

$$\overline{ctx} \vdash x :: \texttt{Zero} \land \overline{ctx} \vdash y \equiv x$$
$$\rightarrow \overline{ctx} \vdash y :: \texttt{Zero}$$

$\overline{a} := \overline{ctx}, a_2 := y :: \texttt{Zero}$
$x := \texttt{"y"}, p_2 = y, p_1 = x$
$a := y :: \texttt{Zero}$
$a_1 := x :: \texttt{Zero}$

Application Comparison

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

$$\begin{aligned} &\overline{ctx} \vdash x :: \texttt{Zero} \wedge \overline{ctx} \vdash y \equiv x && \text{(C-Subst)} \\ &\to \overline{ctx} \vdash y :: \texttt{Zero} \end{aligned}$$

Analyze C-Subst

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

Analyze C-Subst

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

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

Analyze C-Subst

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

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

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

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

C-Subst: Algorithmic Instantiations

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

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

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

C-Subst: Algorithmic Instantiations

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

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

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

C-Subst: Algorithmic Instantiations

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

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

$$
\begin{aligned}
a_2 &:= <input> \\
a &:= a_{2\{\!\{p_2 \mapsto x\}\!\}} \\
a_1 &:= a_{\{\!\{x \mapsto p_1\}\!\}}
\end{aligned}
$$

Operational Semantics

$$\frac{S = \{\langle \overline{a}; e\rangle \mid \langle \overline{a}; e\rangle \in MImpl(m, x) \wedge HC(h) \vdash \overline{a}\} \qquad \langle \overline{a}; e\rangle \in S \\ \forall \langle \overline{a'}; e'\rangle \in S.\ (e' \neq e) \longrightarrow (\overline{a'} \vdash \overline{a}) \wedge \neg(\overline{a} \vdash \overline{a'})}{\langle h; m(x)\rangle \to \langle h; e\rangle}\ \text{R-Call}$$

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

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

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

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

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

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

Operational Semantics: Application

$$h := [x \to (\mathtt{Zero}, nil), \qquad e := \mathtt{prev}(y)$$

$$y \to (\mathtt{Succ}, [(p, x)])] \quad HC(h) := [x.\mathbf{cls} \equiv \mathtt{Zero}, y.\mathbf{cls} \equiv \mathtt{Succ}, y.p \equiv x]$$

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

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

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

Operational Semantics: Application

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

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

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

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

Operational Semantics: Application

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

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

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

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

Type Assignment

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

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

Type Assignment

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

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

Implementation:

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

Type Assignment: Example

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

Types of **new** `Zero`()

Type Assignment: Example

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

Types of **new** `Zero`()

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

Type Assignment: Example

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

Types of **new** `Zero`()

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

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

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

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) == ... && FV(b) == ... &&
    typeassignment(a, e).exists {
      // y, b exists
      ...
    }
}
```

Well-formedness: Example

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

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

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

Context Enrichment

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

Context Enrichment

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

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

Context Enrichment

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

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

$$[x :: C] \vdash y :: C$$

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

Context Enrichment

$$\begin{aligned} \overline{a} &\vdash a \\ \overline{b} +\!\!+\, \overline{a} &\vdash a \end{aligned}$$

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

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

Information Transfer: Compile Time to Runtime

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

Information Transfer: Compile Time to Runtime

$$
\begin{aligned}
e &: [x.\ \overline{a}] \\
\overline{b} &\vdash b && \text{(origin: } \textit{interp}(e)\text{)} \\
\overline{a} +\!\!+\ \overline{b} &\vdash b
\end{aligned}
$$

Information Transfer: Compile Time to Runtime

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

$$\overline{b} \vdash b \qquad \text{(origin: } \mathit{interp}(e))$$

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

$$e : [y.\ y :: \texttt{Nat}]$$

$$e \rightarrow x$$

$$[x :: \texttt{Zero}] \vdash x :: \texttt{Nat}$$

$$[x :: \texttt{Nat}, x :: \texttt{Zero}] \vdash x :: \texttt{Nat}$$

Summary

- Dependent Classes
- DC_C Calculus
- Implementation
  - First-order Model of Sequent Calculus
  - Rule Optimization
  - Interpreter
  - Typechecker
- Information: Compile Time to Runtime