Object Calculus O

Classes are names for which a **method table** is declared. Methods are procedures that take the invoking object (self) as first argument.

Objects are pairs consisting of a class and a **field table**. All values can be fields. If o is an object and l is a label, a **selection** $o.l$ reduces either to the field for o and l or to the application $t\ o$, where t is the method for the class of o and l.

Interfaces are names that are used for typing. Classes and interfaces are collectively referred to as **object types**.

$C \in Cla$	class
$O \in OTy = Cla \uplus Int$	object type
$T \in Ty \ = \ O \ \mid \ T \to T$	type
$l \in Lab$	label
$x \in Var$	variable
$f \in Lab \overset{fin}{\rightharpoonup} Ter$	field table
$t \in Ter \ = \ x \ \mid \ \lambda x{:}T.t \ \mid \ t\,t \ \mid \ obj\ C\ f \ \mid \ t.l$	term
$\Gamma \in Var \rightharpoonup Ty$	type environment

A **procedure** is a closed abstraction $\lambda x{:}T.t$. The set of all procedures is denoted by *Pro*. A **program** consists of two sets *Cla* and *Int* and three further constituents:

1. a **method table** $mth \in Cla \to Lab \overset{fin}{\rightharpoonup} Pro$
2. a **type table** $ty \in OTy \to Lab \overset{fin}{\rightharpoonup} Ty$
3. a **subtype order**, which is a partial order on *OTy*.

The proper reduction rules are as follows:

$(\lambda x{:}T.t)t' \ \to \ t[x := t']$	**beta reduction**
$(obj\ C\ f).l \ \to \ f\ l \quad \text{if } l \in Dom\, f$	**field access**
$(obj\ C\ f).l \ \to \ (mth\ C\ l)(obj\ C\ f) \quad \text{if } l \in Dom\,(mth\ C)$	**method invocation**

The subtype order is extended to all types by $\dfrac{T_1' \le T_1 \qquad T_2 \le T_2'}{T_1 \to T_2 \le T_1' \to T_2'}$.

The typing relations are defined as follows:

$$\Gamma \vdash t : T \quad \overset{\text{def}}{\iff} \quad \exists T' :\ \Gamma \mapsto t : T' \ \wedge \ T' \le T$$

$$\frac{\Gamma x = T}{\Gamma \mapsto x : T} \qquad \frac{\Gamma[x := T] \mapsto t : T'}{\Gamma \mapsto \lambda x{:}T.t : T \to T'} \qquad \frac{\Gamma \mapsto t : T' \to T \quad \Gamma \vdash t' : T'}{\Gamma \mapsto t\,t' : T}$$

$$\frac{\mathit{Dom}\ f = \mathit{Dom}\ (\mathit{ty}\ C) - \mathit{Dom}\ (\mathit{mth}\ C) \qquad \forall l \in \mathit{Dom}\ f{:}\ \ \Gamma \vdash f\,l : \mathit{ty}\ C\ l}{\Gamma \mapsto \mathit{obj}\ C\ f : C}$$

$$\frac{\Gamma \mapsto t : O \quad \mathit{ty}\ O\ l = T}{\Gamma \mapsto t.l : T}$$

A programm is **well-typed** if the following conditions hold:

1. $\mathit{Dom}\ (\mathit{mth}\ C) \subseteq \mathit{Dom}\ (\mathit{ty}\ C)$
2. $t = \mathit{mth}\ C\ l \ \wedge\ T = \mathit{ty}\ C\ l \implies \emptyset \vdash t : C \to T$
3. $C \leq C' \implies \mathit{Dom}\ (\mathit{ty}\ C) \supseteq \mathit{Dom}\ (\mathit{ty}\ C') \ \wedge\ \forall l \in \mathit{Dom}\ (\mathit{ty}\ C'){:}\ \ \mathit{ty}\ C\ l \leq \mathit{ty}\ C'\ l$

For well-typed programs we have the following properties:

- Subsumption and Least Type
- Preservation and Progress
- Confluent reduction

Exercise (Recursion Operators) Let T, T' be types. Write a class *Fix* that provides a recursion operator for $T \to T'$ through a method *fix*.

Type Case

O can be extended with a **type case** as follows:

$t \ = \ \dots \mid \text{case}\ O\ t\ t\ t$

$\text{case}\ O\ (\mathit{obj}\ C\ f)\ t_0\ t_1 \ \to\ t_0\ (\mathit{obj}\ C\ f) \quad \text{if } C \leq O$

$\text{case}\ O\ (\mathit{obj}\ C\ f)\ t_0\ t_1 \ \to\ t_1\ (\mathit{obj}\ C\ f) \quad \text{if not } C \leq O$

$$\frac{\Gamma \vdash t : O_1 \quad \Gamma \mapsto t_0 : O_0 \to T_0 \quad \Gamma \mapsto t_1 : O_1 \to T_1 \quad O \leq O_0 \quad T = \mathit{lub}\ T_0\ T_1}{\Gamma \mapsto \text{case}\ O\ t\ t_0\ t_1 : T}$$

Exercise (Type Tests) Show how a construct t instanceof O that tests whether the object t evaluates to has type O can be expressed with type case. Assume that Boolean values are realized with an interface *Bool* and two subclasses *False* and *True* that have no fields and no methods.

Exercise (Conditionals) Show how a conditional *if* t *then* t_0 *else* t_1 can be expressed with type case.

Example Program

The following program shows how unit, bool and the natural numbers can be implemented in O.

class *Unit*

interface *Bool*
  $if : (Unit \to Nat) \to (Unit \to Nat) \to Nat$

class *False* < *Bool*
  mth $if = \lambda \sigma f g.\ g(obj\ Unit\ \{\})$

class *True* < *Bool*
  mth $if = \lambda \sigma f g.\ f(obj\ Unit\ \{\})$

interface *Nat*
  $pred : Nat$
  $add : Nat \to Nat$
  $isz : Bool$
  $sub : Nat \to Nat$

class *Z* < *Nat*
  mth $pred = \lambda \sigma.\ \sigma$
  mth $add = \lambda \sigma n.\ n$
  mth $isz = \lambda \sigma.\ obj\ True\ \{\}$
  mth $sub = \lambda \sigma n.\ \sigma$

class *P* < *Nat*
  fld $pred : Nat$
  mth $add = \lambda \sigma n.\ \sigma.pred.add\ (obj\ P\ \{pred = n\})$
  mth $isz = \lambda \sigma.\ obj\ False\ \{\}$
  mth $sub = \lambda \sigma n.\ n.isz.if\ (\lambda u.\ \sigma)\ (\lambda u.\ \sigma.pred.sub\ n.pred)$