Featherweight Java

Syntax:

P	$::=$	$\overline{CL}\ e$	program
CL	$::=$	class C extends $C\{\overline{C}\ \overline{f};\ K\ \overline{m}\}$	class definition
K	$::=$	$C(\overline{C}\ \overline{f})\{super(\overline{f});\ this.\overline{f} = \overline{f};\}$	constructor
M	$::=$	$C\ m(\overline{C}\ \overline{x})\{return\ e;\}$	method
e	$::=$		expressions
		x	variables
		$e.f$	field access
		$e.m(\overline{e})$	method invocation
		$new\ C(\overline{e})$	object creation
		$(C)e$	cast

Notes: C is for class names, m is for method names, and f is for field names. This syntax is literally a subset of Java. A Featherweight Java program can be run by a standard Java interpreter.

Subtyping

$$\frac{}{C <: C}$$

$$\frac{C <: D \qquad D <: E}{C <: E}$$

$$\frac{CT(C) = \mathit{class}\ C\ \mathit{extends}\ D\{\ldots\}}{C <: D}$$

Auxiliary Functions

fields(Object) = []
fields(C) =
  let CT(C) = class C extends D $\{\overline{C}\ \overline{f}$; K $\overline{M}\}$
  return fields(D), $\overline{C}\ \overline{f}$

mbody(m,C) =
  let CT(C) = class C extends D $\{\overline{C}\ \overline{f}$; K $\overline{M}\}$
  if $B\ m(\overline{B}\ \overline{x})\{return\ e;\} \in \overline{M}$ **then**
    return $(\overline{x}, e)$
  else
    return mbody(m, D)

More Auxiliary Functions

mtype(m, C) =
  let CT(C) = class C extends D $\{\overline{C}\ \overline{f}$; K $\overline{M}\}$
  if $B\ m(\overline{B}\ \overline{x})\{return\ e;\} \in \overline{M}$ **then**
    return $\overline{B} \rightarrow B$
  else
    return mtype(m, D)

override(m,D,$\overline{C} \rightarrow C_0$) =
  if mtype(m,D) = $\overline{D} \rightarrow D_0$ **then**
    return $\overline{C} = \overline{D}$ **and** $C_0 = D_0$
  else
    return true

Evaluation

Values:

$$v ::= new\ C(\overline{v})$$

Evaluation contexts:

$$\begin{array}{rcl} E & ::= & [] \mid E.f \mid E.m(\overline{e}) \mid v.m(\overline{v}, E, \overline{e}) \mid \\ & & new\ C(\overline{v}, E, \overline{e}) \mid (C)E \end{array}$$

Reduction rules:

$$\text{(E-Proj)}\ \frac{fields(C) = \overline{C}\ \overline{f}}{new\ C(\overline{v}).f_i \longrightarrow v_i}$$

$$\text{(E-Invk)}\ \frac{mbody(m, C) = (\overline{x}, e)}{(new\ C(\overline{v})).m(\overline{u}) \longrightarrow [\overline{x} \mapsto \overline{u}, this \mapsto new\ C(\overline{v})]e}$$

$$\text{(E-Cast)}\ \frac{C <: D}{(D)(new\ C(\overline{v})) \longrightarrow new\ C(\overline{v})}$$

Type Rules

$\boxed{\Gamma \vdash e : C}$

$$\text{(T-Var)}\frac{x : C \in \Gamma}{\Gamma \vdash x : C} \qquad \text{(T-Field)}\frac{\Gamma \vdash e : C \qquad \mathit{fields}(C) = \overline{C}\ \overline{f}}{\Gamma \vdash e.f_i : C_i}$$

$$\text{(T-Invk)}\frac{\Gamma \vdash e : C \qquad \mathit{mtype}(m, C) = \overline{D} \to C_r \qquad \overline{C} <: \overline{D}}{\Gamma \vdash e.m(\overline{e}) : C_r}$$

$$\text{(T-New)}\frac{\mathit{fields}(C) = \overline{D}\ \overline{f} \qquad \Gamma \vdash \overline{e} : \overline{C} \qquad \overline{C} <: \overline{D}}{\Gamma \vdash \mathit{new}\ C(\overline{e}) : C}$$

$$\text{(T-Cast)}\frac{\Gamma \vdash e : D \qquad C <: D \text{ or } D <: C}{\Gamma \vdash (C)e : C}$$

Type Rules for Methods and Classes

$\boxed{M \ OK \ in \ C}$

$$\frac{\overline{x} : \overline{C}, this : C \vdash e : C_e \qquad C_e <: C_r \quad CT(C) = class \ C \ extends \ D\{\ldots\} \quad override(m, D, \overline{C} \rightarrow C_r)}{C_r \ m(\overline{C} \ \overline{x})\{return \ e;\} \ OK \ in \ C}$$

$\boxed{C \ OK}$

$$\frac{K = C(\overline{D} \ \overline{g}, \overline{C} \ \overline{f})\{super(\overline{g}); this.\overline{f} = \overline{f};\} \quad fields(D) = \overline{D} \ \overline{g} \qquad \overline{M} \ OK \ in \ C}{class \ C \ extends \ D\{\overline{C} \ \overline{f}; K \ \overline{M}\} \ OK}$$

Properties

Lemma (Preservation)

If $\Gamma \vdash e : C$ *and* $e \longmapsto e'$ *then* $\Gamma \vdash e' : C'$ *for some* D *where* $C' <: C$.

Definition

A **bad cast** is an expression of the form $(C)(new\ D(\overline{v}))$ where $D \not<: C$.

Lemma (Progress)

Suppose e *is closed, well-typed, and in normal form. Then either* e *is a value or there is an evaluation context* E *such that* $e = E[b]$ *where* b *is a bad cast.*