Chapter 19: Case Study: Featherweight Java

Syntax
Typing
Evaluation
Properties
What is Object-Oriented Programming

- **Multiple representations**
  - Object (instances)

- **Encapsulation**
  - Internal representation/implementation is hidden

- **Subtyping**
  - Object interface

- **Inheritance**
  - Class, subclass, superclass

- **Open recursion.**
  - Self (this)

Chapter 19: direct treatment (treat objects as primitive) of a core object-oriented language based on Java (rather than encoding the features in lambda-calculus with subtyping, records, and references in Chapter 18.)
FJ: Featherweight Java

• Proposed by Igarashi, Pierce, and Wadler (1999)
• A minimal core calculus for modeling Java’s type system
• The goal in designing FJ was to make its proof of type safety as concise as possible, while still capturing the essence of the safety argument for the central features of full Java.

We used FJ in our paper:
An FJ Program

class A extends Object { A() { super(); } }

class B extends Object { B() { super(); } }

class Pair extends Object {
   Object fst;
   Object snd;
   // Constructor:
   Pair(Object fst, Object snd) {
      super(); this.fst=fst; this.snd=snd; }
   // Method definition:
   Pair setfst(Object newfst) {
      return new Pair(newfst, this.snd); }
}

((Pair) (new Pair(new Pair(new A(),new B()), new A()).fst).snd}
Nominal and Structural Type Systems

- Type names: fundamental stylistic difference between FJ (and Java) and the typed lambda-calculi.

  \[
  \text{NatPair} = \{\text{fst: Nat, snd: Nat}\};
  \]

  ➢ Nominal type systems:
  - Types are always named.
  - Typechecker mostly manipulates names, not structures.
  - Subtyping is declared explicitly by programmer.

  ➢ Structural type systems:
  - What matters about a type (for typing, subtyping, etc.) is just its structure.
  - Names are just convenient (but inessential) abbreviations.
Syntax

CL ::= class declarations:
    class C extends C { C f; K M }

K ::= constructor declarations:
    C(C f) { super(f); this.f=f; }

M ::= method declarations:
    C m(C x) { return t; }

| t | ::= terms:
    x variable
    t.f field access
    t.m(t) method invocation
    new C(t) object creation
    (C) t cast

v ::= values:
    new C(v) object creation
Subtyping

\[
\begin{align*}
C & \lla C \\
C & \lla D \quad D \lla E \\
\hline
C & \lla E \\
\hline
CT(C) = \text{class } C \text{ extends } D \{\ldots\} \\
\hline
C & \lla D
\end{align*}
\]
Auxiliary Functions

Field lookup
\[
\text{fields}(\text{Object}) = \bullet
\]
\[
CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \}
\]
\[
\text{fields}(D) = \bar{D} \bar{g}
\]
\[
\text{fields}(C) = \bar{D} \bar{g}, \bar{C} \bar{f}
\]

Method type lookup
\[
\text{mtype}(m, C) = \bar{C} \rightarrow \bar{C}
\]
\[
CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \}
\]
\[
B m (B x) \{ \text{return } t ; \} \in \bar{M}
\]
\[
\text{mtype}(m, C) = B \rightarrow B
\]
\[
CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \}
\]
\[
m \text{ is not defined in } \bar{M}
\]
\[
\text{mtype}(m, C) = \text{mtype}(m, D)
\]

Method body lookup
\[
\text{mbody}(m, C) = (\bar{x}, t)
\]
\[
CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \}
\]
\[
B m (B x) \{ \text{return } t ; \} \in \bar{M}
\]
\[
\text{mbody}(m, C) = (\bar{x}, t)
\]
\[
CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \}
\]
\[
m \text{ is not defined in } \bar{M}
\]
\[
\text{mbody}(m, C) = \text{mbody}(m, D)
\]

Valid method overriding
\[
\text{override}(m, D, \bar{C} \rightarrow C_0)
\]
\[
\text{mtype}(m, D) = D \rightarrow D_0 \text{ implies } \bar{C} = \bar{D} \text{ and } C_0 = D_0
\]
\[
\text{override}(m, D, \bar{C} \rightarrow C_0)
\]
Evaluation

\[
\begin{align*}
\text{fields}(C) &= \overline{C} \overline{f} \\
(new \ C(V)).f_i &\rightarrow v_i \quad \text{(E-PROJNEW)} \\
mbody(m, C) &= (\overline{x}, t_0) \\
(new \ C(V)).m(\overline{u}) &\rightarrow [\overline{x} \mapsto \overline{u}, \text{this} \mapsto \text{new} \ C(V)]t_0 \\
C &<: D \\
(D)(new \ C(V)) &\rightarrow \text{new} \ C(V) \quad \text{(E-CASTNEW)} \\
t_0 &\rightarrow t_0' \\
t_0 . f &\rightarrow t_0' . f \quad \text{(E-FIELD)} \\
\frac{t_0 \rightarrow t_0'}{t_0 . m(t) \rightarrow t_0' . m(t)} \quad \text{(E-INVK-RECV)} \\
\frac{t_i \rightarrow t_i'}{\overline{v}_0 . m(\overline{v}, t_i, \overline{t}) \rightarrow \overline{v}_0 . m(\overline{v}, t_i', \overline{t})} \quad \text{(E-INVK-ARG)} \\
\frac{t_i \rightarrow t_i'}{\text{new} \ C(\overline{v}, t_i, \overline{t}) \rightarrow \text{new} \ C(\overline{v}, t_i', \overline{t})} \quad \text{(E-NEW-ARG)} \\
\frac{t_0 \rightarrow t_0'}{(C)t_0 \rightarrow (C)t_0'} \quad \text{(E-CAST)}
\end{align*}
\]
### Typing

**Term typing**

\[
\frac{x : C \in \Gamma}{\Gamma \vdash x : C} \quad \text{(T-VAR)}
\]

\[
\frac{\Gamma \vdash t_0 : C_0 \quad \text{fields}(C_0) = \overline{C \vec{f}}}{\Gamma \vdash t_0.f_i : C_i} \quad \text{(T-FIELD)}
\]

\[
\frac{\Gamma \vdash t_0 : C_0 \quad mtype(m, C_0) = D \rightarrow C}{\Gamma \vdash t_0.m(\overline{t}) : C} \quad \text{(T-INVK)}
\]

\[
\frac{\text{fields}(C) = \overline{D \vec{f}}}{\Gamma \vdash t : \overline{C} \quad \overline{C} \triangleleft D} \quad \text{(T-NEW)}
\]

**Method typing**

\[
\frac{\Gamma \vdash t_0 : D \quad C \triangleleft D \quad D \neq C}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-UCAST)}
\]

**Class typing**

\[
\frac{\Gamma \vdash t_0 : D \quad D \triangleleft C}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-UCAST)}
\]

\[
\frac{\Gamma \vdash (C)t_0 : C \quad mtype(m, C_0) = D \rightarrow C}{\Gamma \vdash t_0 : E_0 \quad E_0 \triangleleft C_0}{\Gamma \vdash \text{override}(m, D, \overline{C \rightarrow C_0})}{\frac{\text{CT}(C) = \text{class } C \text{ extends } D \{\ldots\}}{C_0 \ m \ (C \ x) \ \{\text{return } t_0;\} \quad \text{OK in } C} \quad \text{(T-UCAST)}}
\]

\[
\frac{\text{fields}(D) = \overline{D \vec{g}} 
\quad \text{M OK in } C \quad \text{super}(g); \ this.f = \overline{f};}{\text{class } C \text{ extends } D \{\overline{C \ f}; \ K \overline{M}\} \quad \text{OK}}
\]
Properties

**Theorem [Preservation]:** If $\Gamma \vdash t : C$ and $t \rightarrow t'$, then $\Gamma \vdash t' : C'$ for some $C' \ll C$.

**Theorem [Progress]:** Suppose $t$ is a closed, well-typed normal form. Then either (1) $t$ is a value, or (2) for some evaluation context $E$, we can express $t$ as $t = E[(\langle C \rangle (\text{new } D(\bar{v})))$, with $D \not\ll C$.

$$
E \ ::= \\
[ ] \\
E.f \\
E.m(t) \\
v.m(\bar{v},E,\bar{t}) \\
\text{new } C(\bar{v},E,\bar{t}) \\
(\langle C \rangle)E
$$
Homework

18.11.1 Exercise [Recommended, ★★★]: Use the fullref checker to implement the following extensions to the classes above:

1. Rewrite instrCounterClass so that it also counts calls to get.

2. Extend your modified instrCounterClass with a subclass that adds a reset method, as in §18.4.

3. Add another subclass that also supports backups, as in §18.7.

Please submit electronically.