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:
Jun Li, Chenglong Wang, Yingfei Xiong, Zhenjiang Hu, SWIN: Towards Type-Safe Java Program Adaptation between APIs, ACM SIGPLAN 2015 Workshop on Partial Evaluation and Program Manipulation (PEPM 2015), Mumbai, India, January 13-14, 2015. pp.91-102.

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.

```
NatPair = {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

Syntax

CL	::=		*class declarations:*
		$\texttt{class C extends C \{}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{; K }\overline{\texttt{M}}\texttt{\}}$	
K	::=		*constructor declarations:*
		$\texttt{C(}\overline{\texttt{C}}\ \overline{\texttt{f}}\texttt{) \{super(}\overline{\texttt{f}}\texttt{); this.}\overline{\texttt{f}}\texttt{=}\overline{\texttt{f}}\texttt{;\}}$	
M	::=		*method declarations:*
		$\texttt{C m(}\overline{\texttt{C}}\ \overline{\texttt{x}}\texttt{) \{return t;\}}$	
t	::=		*terms:*
		$\texttt{x}$	*variable*
		$\texttt{t.f}$	*field access*
		$\texttt{t.m(}\overline{\texttt{t}}\texttt{)}$	*method invocation*
		$\texttt{new C(}\overline{\texttt{t}}\texttt{)}$	*object creation*
		$\texttt{(C) t}$	*cast*
v	::=		*values:*
		$\texttt{new C(}\overline{\texttt{v}}\texttt{)}$	*object creation*

Subtyping

Subtyping $\boxed{\texttt{C<:D}}$

$$\texttt{C} <: \texttt{C}$$

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

$$\frac{CT(\texttt{C}) = \texttt{class C extends D \{...\}}}{\texttt{C} <: \texttt{D}}$$

Auxiliary Functions

Field lookup $\boxed{fields(\mathtt{C}) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}$

$$fields(\mathtt{Object}) = \bullet$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \qquad fields(\mathtt{D}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}}}{fields(\mathtt{C}) = \overline{\mathtt{D}}\ \overline{\mathtt{g}}, \overline{\mathtt{C}}\ \overline{\mathtt{f}}}$$

Method type lookup $\boxed{mtype(\mathtt{m},\mathtt{C}) = \overline{\mathtt{C}}\to\mathtt{C}}$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathtt{B\ m\ (\overline{B}\ \overline{x})\ \{return\ t;\}} \in \overline{\mathtt{M}}}{mtype(\mathtt{m},\mathtt{C}) = \overline{\mathtt{B}}\to\mathtt{B}}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathtt{m} \text{ is not defined in } \overline{\mathtt{M}}}{mtype(\mathtt{m},\mathtt{C}) = mtype(\mathtt{m},\mathtt{D})}$$

Method body lookup $\boxed{mbody(\mathtt{m},\mathtt{C}) = (\overline{\mathtt{x}},\mathtt{t})}$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathtt{B\ m\ (\overline{B}\ \overline{x})\ \{return\ t;\}} \in \overline{\mathtt{M}}}{mbody(\mathtt{m},\mathtt{C}) = (\overline{\mathtt{x}},\mathtt{t})}$$

$$\frac{CT(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\}} \qquad \mathtt{m} \text{ is not defined in } \overline{\mathtt{M}}}{mbody(\mathtt{m},\mathtt{C}) = mbody(\mathtt{m},\mathtt{D})}$$

Valid method overriding $\boxed{override(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}}\to\mathtt{C}_0)}$

$$\frac{mtype(\mathtt{m},\mathtt{D}) = \overline{\mathtt{D}}\to\mathtt{D}_0 \text{ implies } \overline{\mathtt{C}} = \overline{\mathtt{D}} \text{ and } \mathtt{C}_0 = \mathtt{D}_0}{override(\mathtt{m}, \mathtt{D}, \overline{\mathtt{C}}\to\mathtt{C}_0)}$$

Evaluation

Evaluation $\boxed{t \longrightarrow t'}$

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

$$\frac{mbody(m, C) = (\overline{x}, t_0)}{\begin{array}{c}(\texttt{new}\ C(\overline{v})).m(\overline{u}) \\ \longrightarrow [\overline{x} \mapsto \overline{u}, \texttt{this} \mapsto \texttt{new}\ C(\overline{v})]t_0\end{array}} \quad \text{(E-InvkNew)}$$

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

$$\frac{t_0 \longrightarrow t_0'}{t_0.f \longrightarrow t_0'.f} \quad \text{(E-Field)}$$

$$\frac{t_0 \longrightarrow t_0'}{t_0.m(\overline{t}) \longrightarrow t_0'.m(\overline{t})} \quad \text{(E-Invk-Recv)}$$

$$\frac{t_i \longrightarrow t_i'}{\begin{array}{c}v_0.m(\overline{v}, t_i, \overline{t}) \\ \longrightarrow v_0.m(\overline{v}, t_i', \overline{t})\end{array}} \quad \text{(E-Invk-Arg)}$$

$$\frac{t_i \longrightarrow t_i'}{\begin{array}{c}\texttt{new}\ C(\overline{v}, t_i, \overline{t}) \\ \longrightarrow \texttt{new}\ C(\overline{v}, t_i', \overline{t})\end{array}} \quad \text{(E-New-Arg)}$$

$$\frac{t_0 \longrightarrow t_0'}{(C)t_0 \longrightarrow (C)t_0'} \quad \text{(E-Cast)}$$

Typing

Term typing $\boxed{\Gamma \vdash t : C}$

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

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

$$\frac{\begin{array}{c}\Gamma \vdash t_0 : C_0 \\ \textit{mtype}(m, C_0) = \overline{D} \to C \\ \Gamma \vdash \overline{t} : \overline{C} \qquad \overline{C} <: \overline{D}\end{array}}{\Gamma \vdash t_0.m(\overline{t}) : C} \quad \text{(T-INVK)}$$

$$\frac{\begin{array}{c}\textit{fields}(C) = \overline{D}\ \overline{f} \\ \Gamma \vdash \overline{t} : \overline{C} \qquad \overline{C} <: \overline{D}\end{array}}{\Gamma \vdash \text{new } C(\overline{t}) : C} \quad \text{(T-NEW)}$$

$$\frac{\Gamma \vdash t_0 : D \qquad D <: C}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-UCAST)}$$

$$\frac{\Gamma \vdash t_0 : D \qquad C <: D \qquad C \neq D}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-DCAST)}$$

$$\frac{\begin{array}{c}\Gamma \vdash t_0 : D \qquad C \not<: D \qquad D \not<: C \\ \textit{stupid warning}\end{array}}{\Gamma \vdash (C)t_0 : C} \quad \text{(T-SCAST)}$$

Method typing $\boxed{M \text{ OK in } C}$

$$\frac{\begin{array}{c}\overline{x} : \overline{C}, \text{this} : C \vdash t_0 : E_0 \qquad E_0 <: C_0 \\ \textit{CT}(C) = \text{class } C \text{ extends } D \{\ldots\} \\ \textit{override}(m, D, \overline{C} \to C_0)\end{array}}{C_0\ m\ (\overline{C}\ \overline{x})\ \{\text{return } t_0;\} \text{ OK in } C}$$

Class typing $\boxed{C \text{ OK}}$

$$\frac{\begin{array}{c}K = C(\overline{D}\ \overline{g}, \overline{C}\ \overline{f}) \\ \{\text{super}(\overline{g});\ \text{this}.\overline{f} = \overline{f};\} \\ \textit{fields}(D) = \overline{D}\ \overline{g} \qquad \overline{M} \text{ OK in } C\end{array}}{\text{class } C \text{ extends } D\ \{\overline{C}\ \overline{f};\ K\ \overline{M}\} \text{ OK}}$$

Properties

THEOREM [PRESERVATION]: If $\Gamma \vdash \texttt{t} : \texttt{C}$ and $\texttt{t} \longrightarrow \texttt{t}'$, then $\Gamma \vdash \texttt{t}' : \texttt{C}'$ for some $\texttt{C}' <: \texttt{C}$. □

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 $\texttt{t} = E[\,(\texttt{C})\,(\texttt{new D}(\overline{\texttt{v}}))\,]$, with $\texttt{D} \not<: \texttt{C}$. □

$$
\begin{array}{lll}
E & ::= & \\
 & & [\,] \\
 & & E\texttt{.f} \\
 & & E\texttt{.m}(\overline{\texttt{t}}) \\
 & & \texttt{v.m}(\overline{\texttt{v}}, E, \overline{\texttt{t}}) \\
 & & \texttt{new C}(\overline{\texttt{v}}, E, \overline{\texttt{t}}) \\
 & & (\texttt{C})E
\end{array}
$$

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.