

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:*
class C extends C { \bar{C} \bar{f} ; K \bar{M} }

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

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

t_i ::= *terms:*
x *variable*
t.f *field access*
t.m(\bar{t}) *method invocation*
new C(\bar{t}) *object creation*
(C) t *cast*

v ::= *values:*
new C(\bar{v}) *object creation*



Subtyping

Subtyping

$C <: D$

$C <: C$

$C <: D \quad D <: E$

$C <: E$

$CT(C) = \text{class } C \text{ extends } D \{ \dots \}$

$C <: D$



Auxiliary Functions

Field lookup

$$\boxed{fields(C) = \bar{C} \bar{f}}$$

$$fields(Object) = \bullet$$

$$\frac{CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \} \quad fields(D) = \bar{D} \bar{g}}{fields(C) = \bar{D} \bar{g}, \bar{C} \bar{f}}$$

Method type lookup

$$\boxed{mtype(m, C) = \bar{C} \rightarrow C}$$

$$\frac{CT(C) = \text{class } C \text{ extends } D \{ \bar{C} \bar{f}; K \bar{M} \} \quad B \ m \ (\bar{B} \ \bar{x}) \ \{ \text{return } t; \} \in \bar{M}}{mtype(m, C) = \bar{B} \rightarrow B}$$

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

Method body lookup

$$\boxed{mbody(m, C) = (\bar{x}, t)}$$

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

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

Valid method overriding

$$\boxed{override(m, D, \bar{C} \rightarrow C_0)}$$

$$\frac{mtype(m, D) = \bar{D} \rightarrow D_0 \text{ implies } \bar{C} = \bar{D} \text{ and } C_0 = D_0}{override(m, D, \bar{C} \rightarrow C_0)}$$



Evaluation

Evaluation

$t \rightarrow t'$

$$\begin{array}{c}
 \frac{fields(C) = \bar{C} \bar{f}}{(new\ C(\bar{v})) . f_i \rightarrow v_i} \quad (E-PROJNEW) \\
 \frac{mbody(m, C) = (\bar{x}, t_0)}{(new\ C(\bar{v})) . m(\bar{u}) \rightarrow [\bar{x} \mapsto \bar{u}, this \mapsto new\ C(\bar{v})] t_0} \quad (E-INVKNOW) \\
 \frac{C <: D}{(D) (new\ C(\bar{v})) \rightarrow new\ C(\bar{v})} \quad (E-CASTNEW) \\
 \frac{t_0 \rightarrow t'_0}{t_0 . f \rightarrow t'_0 . f} \quad (E-FIELD)
 \end{array}$$

$$\begin{array}{c}
 \frac{t_0 \rightarrow t'_0}{t_0 . m(\bar{t}) \rightarrow t'_0 . m(\bar{t})} \quad (E-INVK-RECV) \\
 \frac{t_i \rightarrow t'_i}{v_0 . m(\bar{v}, t_i, \bar{t}) \rightarrow v_0 . m(\bar{v}, t'_i, \bar{t})} \quad (E-INVK-ARG) \\
 \frac{t_i \rightarrow t'_i}{new\ C(\bar{v}, t_i, \bar{t}) \rightarrow new\ C(\bar{v}, t'_i, \bar{t})} \quad (E-NEW-ARG) \\
 \frac{t_0 \rightarrow t'_0}{(C) t_0 \rightarrow (C) t'_0} \quad (E-CAST)
 \end{array}$$



Typing

Term typing

$\Gamma \vdash t : C$

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

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

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

$$\frac{\text{fields}(C) = \bar{D} \bar{f} \quad \Gamma \vdash \bar{t} : \bar{C} \quad \bar{C} <: \bar{D}}{\Gamma \vdash \text{new } C(\bar{t}) : C} \quad (\text{T-NEW})$$

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

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

$$\frac{\Gamma \vdash t_0 : D \quad C \not<: D \quad D \not<: C \quad \text{stupid warning}}{\Gamma \vdash (C)t_0 : C} \quad (\text{T-SCAST})$$

Method typing

$\text{M OK in } C$

$$\frac{\bar{x} : \bar{C}, \text{this} : C \vdash t_0 : E_0 \quad E_0 <: C_0 \quad CT(C) = \text{class } C \text{ extends } D \{ \dots \} \quad \text{override}(m, D, \bar{C} \rightarrow C_0)}{C_0 \text{ m } (\bar{C} \bar{x}) \{ \text{return } t_0; \} \text{ OK in } C}$$

Class typing

$C \text{ OK}$

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



Properties

THEOREM [PRESERVATION]: If $\Gamma \vdash t : C$ and $t \longrightarrow t'$, then $\Gamma \vdash t' : C'$ for some $C' \leq C$. \square

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 $t = E[(C)(\text{new } D(\bar{v}))]$, with $D \not\leq C$. \square

$$\begin{aligned} E ::= & \\ & [] \\ & E.f \\ & E.m(\bar{t}) \\ & v.m(\bar{v}, E, \bar{t}) \\ & \text{new } C(\bar{v}, E, \bar{t}) \\ & (C)E \end{aligned}$$


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.

