Featherweight Java, FJ

Type Systems Course,

Spring 2012, Computer Science dept., Lund University

Amr Ergawy

23 May 2012

Outline

- References.
- Prerequisites and Discussion Scope.
- FJ: What and Why?
- Type systems: Nominal vs. Structural.
- FJ details: syntax, evaluation, and typing.
- FJ soundness: progress and preservation.
- An extension: Generic FJ.
- Optional disucssion: detecting logical type errors with FJ?
- Suggested excercises.

References

1. Types and Programming Languages, Benjamin C. Pierce.
2. Slides from week 13 of the course Software foundations course given at EPFL by Martin Odersky.
3. Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. 2001. Featherweight Java: a minimal core calculus for Java and GJ. *ACM Trans. Program. Lang. Syst.* 23, 3 (May 2001), 396-450. DOI=10.1145/503502.503505 http://doi.acm.org/10.1145/503502.503505.

Prerequisites and Discussion Scope

- From the course book [1], discussing FJ depends on these topics:
  - Induction, grammars, semantics, evaluations, derviations, and typed BN.
  - The typing relation, λ-Calculus, typed λ-Calculus, simple extensions of λ-Calculus, and subtyping.
- In the course book [1], these topics are discussed in chapters 2, 3, 5, 8, 9, 11, 15.
- Execuse me! I can not engage in discussions outside this scope! ☹

FJ: What and Why? 1/2

- FJ is a compact formal model that enables studying properties of Java and its extensions.
- Formally:
  - FJ is a subset of Java.
  - This subset includes key features of Java.
  - This subset excludes complex features of Java.
- As a result:
  - We can apply well-understood theory to this subset, i.e. FJ.
  - Incrementally, we can apply the same theory to extensions, e.g. Generic FJ and Feature FJ.
- Every FJ program is a functional Java Program.

FJ: What and Why? 2/2

- Applying the theory of type-safety on FJ focuses on these central features:
  - Mutually recursive class definition, Subtyping.
  - Object creation, Casting.
  - Field access.
  - Method invocation, override, recursion through “this” keyword.
- Applying the theory of type-safety on FJ excludes these features:
  - Assignment, interfaces, overloading, using “super” keyword, null pointers, base types, abstract methods, inner classes, shadowing super class fields, access control, and exceptions
  - Other more advanced features, e.g. concurrency and reflection.

Example FJ Program 1/3

- Omitting the assignment defines pure functional FJ:
  - Objects only initialized via constructors.
  - No state modifications.
- “=” and “super” only appear in a constructor.
- The method setfst() returns a new object, but does not modify the existing one.

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

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

class Pair extends Object {
  Object fst;
  Object snd;

  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd; }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd); }
}
```

Copied from reference [2]

Example FJ Program 2/3

- In a constructor, the order of parameters is the same as the order of the class fields:
  - Accessing a field is just selecting the constructor parameter that has the same order.

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

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

class Pair extends Object {
  Object fst;
  Object snd;

  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd; }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd); }
}
```

Copied from reference [2]

Example FJ Program 3/3

- A value in FJ is a "new"-term.
- In the course book [1], FJ uses "call-by-value". The reference paper [3] has another assumption: non-deterministic beta reduction relation ☹.
- A method invocation looks up the method in the "new"-term that exists before its preceding '.':
  - Enabling method overriding.
- A method invocation substitutes its parameters:
  - "this" is considered a variable and is substituted by the object itself.
  - Enabling recursion through "self".

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

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

class Pair extends Object {
  Object fst;
  Object snd;

  Pair(Object fst, Object snd) {
    super(); this.fst=fst; this.snd=snd; }

  Pair setfst(Object newfst) {
    return new Pair(newfst, this.snd); }
}
```

Copied from reference [2]

Type systems: Nominal vs. Structural. 1/2

- Nominal type systems:
  - Type names are essential, e.g. used in a class table.
  - Subtyping is explicitly declared.
- Structural type systems:
  - type structures may be used in exchange with type names.
  - Subtyping is directly defined on the structures of types, e.g. T-RcdWidth on records subtyping, Chapter 11 in the course book [1].
- “Nominal vs. Structural” is a continuous discussion in the research community.
- In this presentation, we focus on the advantages of nominal type system, as it is used by FJ.

Type systems: Nominal vs. Structural. 2/2

- Advantages of nominal type systems:
  - Easy run-time type tests, e.g. using "*instanceof*".
  - It is simple to support:
    - Recursive types.
    - Mutually recursive types.
  - It is required to only check for once that a type is "structurally" a sub-type of another that is explicitly declared its "super" type. Then subtyping is checked from a type table.
  - It prevents "spurious* subsumption" : a compiler "structurally" accepts a subtyping relation of two completely unrelated types.

* *spurious: fake, unreal*

FJ Syntax 1/5

- A dashed term or class symbol: a list of ‘,’ separated terms or class symbols.
- A dashed term or class symbol followed by ‘;’: a list of ‘;’ separated terms or class symbols.
- A value in FJ is a “new”-term.

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

Copied from reference [2]

FJ Syntax 2/5

- In the method declaration, M, $\bar{x}$ is bound in t.

K ::=		*constructor declarations*
	$\texttt{C}(\bar{\texttt{C}}\ \bar{\texttt{f}})\ \{\texttt{super}(\bar{\texttt{f}});\ \texttt{this}.\bar{\texttt{f}}=\bar{\texttt{f}};\}$	
M ::=		*method declarations*
	$\texttt{C m}(\bar{\texttt{C}}\ \bar{\texttt{x}})\ \{\texttt{return t};\}$	
CL ::=		*class declarations*
	$\texttt{class C extends C}\ \{\bar{\texttt{C}}\ \bar{\texttt{f}};\ \texttt{K}\ \bar{\texttt{M}}\}$	

Copied from reference [2]

FJ Syntax 3/5

- CT is the ”class table”.
- For sanity conditions of CT, see page 256 of the course book [1].
- Because of the CT sanity and explicit declaration of inheritance, i.e. subtyping:
  - class Object is assured to be the “top” of the class hierarchy.
  - No need for a “Top” rule.
  - The book answer to exercise 19.4.1.

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

$$\mathtt{C} <: \mathtt{C}$$

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

Copied from reference [2]

FJ Syntax 4/5

- Lookups: fields, method type, and method body.
- Note the role of the super class.

$$\mathit{fields}(\mathtt{Object}) = \emptyset$$

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

Copied from reference [2]

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

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

Copied from reference [2]

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

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

Copied from reference [2]

FJ Syntax 5/5

- To override a method of a super class in a subclass, we must keep the same parameters and return types.

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

Copied from reference [2]

FJ Evaluation 1/2

- In these computation evaluation rules, a non-well typed ”program” stucks when:
  - Attempting to access a field not in a class: field lookup fails.
  - Attempting to invoke a method not in a class: method lookup failes.
  - Solution: compiler’s job?
- A well-typed program may stuck in evaluation using E-CastNew if C is not a subclass of D:
  - In the course book [1], chapter 15, page 195, run-time check is suggested to recover preservation. One suggest recovering mechanism is raising exceptions.
  - In chapter 19, exercise 19.4.4 suggests following chapter 14 to extend FJ with exceptions.

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

$$\frac{mbody(\mathtt{m}, \mathtt{C}) = (\overline{\mathtt{x}}, \mathtt{t}_0)}{(\mathtt{new}\ \mathtt{C}(\overline{\mathtt{v}})).\mathtt{m}(\overline{\mathtt{u}}) \longrightarrow [\overline{\mathtt{x}} \mapsto \overline{\mathtt{u}}, \mathtt{this} \mapsto \mathtt{new}\ \mathtt{C}(\overline{\mathtt{v}})]\mathtt{t}_0} \quad \text{(E-InvkNew)}$$

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

Copied from reference [2]

FJ Evaluation 2/2

- Congruence rules:

$$\frac{\mathtt{t}_0 \longrightarrow \mathtt{t}_0'}{\mathtt{t}_0\mathtt{.f} \longrightarrow \mathtt{t}_0'\mathtt{.f}} \qquad \text{(E-Field)}$$

$$\frac{\mathtt{t}_0 \longrightarrow \mathtt{t}_0'}{\mathtt{t}_0\mathtt{.m(}\overline{\mathtt{t}}\mathtt{)} \longrightarrow \mathtt{t}_0'\mathtt{.m(}\overline{\mathtt{t}}\mathtt{)}} \qquad \text{(E-Invk-Recv)}$$

$$\frac{\mathtt{t}_i \longrightarrow \mathtt{t}_i'}{\mathtt{v}_0\mathtt{.m(}\overline{\mathtt{v}}\mathtt{,\ t}_i\mathtt{,\ }\overline{\mathtt{t}}\mathtt{)} \longrightarrow \mathtt{v}_0\mathtt{.m(}\overline{\mathtt{v}}\mathtt{,\ t}_i'\mathtt{,\ }\overline{\mathtt{t}}\mathtt{)}} \qquad \text{(E-Invk-Arg)}$$

$$\frac{\mathtt{t}_i \longrightarrow \mathtt{t}_i'}{\mathtt{new\ C(}\overline{\mathtt{v}}\mathtt{,\ t}_i\mathtt{,\ }\overline{\mathtt{t}}\mathtt{)} \longrightarrow \mathtt{new\ C(}\overline{\mathtt{v}}\mathtt{,\ t}_i'\mathtt{,\ }\overline{\mathtt{t}}\mathtt{)}} \qquad \text{(E-New-Arg)}$$

$$\frac{\mathtt{t}_0 \longrightarrow \mathtt{t}_0'}{\mathtt{(C)t}_0 \longrightarrow \mathtt{(C)t}_0'} \qquad \text{(E-Cast)}$$

Copied from reference [2]

FJ Typing 1/3

$$\frac{\mathtt{x}:\mathtt{C} \in \Gamma}{\Gamma \vdash \mathtt{x} : \mathtt{C}} \qquad \text{(T-Var)}$$

$$\frac{\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \qquad \mathit{fields}(\mathtt{C}_0) = \overline{\mathtt{C}}\ \overline{\mathtt{f}}}{\Gamma \vdash \mathtt{t}_0.\mathtt{f}_i : \mathtt{C}_i} \qquad \text{(T-Field)}$$

$$\frac{\begin{array}{c}\Gamma \vdash \mathtt{t}_0 : \mathtt{C}_0 \\ \mathit{mtype}(\mathtt{m}, \mathtt{C}_0) = \overline{\mathtt{D}} \to \mathtt{C} \\ \Gamma \vdash \overline{\mathtt{t}} : \overline{\mathtt{C}} \qquad \overline{\mathtt{C}} <: \overline{\mathtt{D}}\end{array}}{\Gamma \vdash \mathtt{t}_0.\mathtt{m}(\overline{\mathtt{t}}) : \mathtt{C}} \qquad \text{(T-Invk)}$$

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

Copied from reference [2]

FJ Typing 2/3

- (A)(Object)newB() has two halves that both seem well typed:
  - one down-cast and another up-cast
  - but calling by value evaluates the term to a casting from a type that has no relation to the casting type.
- We evaluated from a well-type term to an ill-typed, breaking preservation.
- Just because such term is possible to exist:
  - T-Scast exists so that type preservation proofs passes failed casting.
  - A type checker produces ”stupid warning” if it uses that rule.
- A question: to make real use of T-Dcast, do not we need to add a rule called: ”E-DownCastNew” and a run-time type check like ”instanceof”?

$$\texttt{(A)}\underline{\texttt{(Object)new B()}} \longrightarrow \texttt{(A)new B()}$$

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{D} \qquad \texttt{C} \not<: \texttt{D} \qquad \texttt{D} \not<: \texttt{C} \qquad \textit{stupid warning}}{\Gamma \vdash \texttt{(C)t}_0 : \texttt{C}} \quad \text{(T-SCAST)}$$

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{D} \qquad \texttt{D} <: \texttt{C}}{\Gamma \vdash \texttt{(C)t}_0 : \texttt{C}} \quad \text{(T-UCAST)}$$

$$\frac{\Gamma \vdash \texttt{t}_0 : \texttt{D} \qquad \texttt{C} <: \texttt{D} \qquad \texttt{C} \neq \texttt{D}}{\Gamma \vdash \texttt{(C)t}_0 : \texttt{C}} \quad \text{(T-DCAST)}$$

Copied from reference [2]

FJ Typing 3/3

- In ”M Ok in C”, t_0, the body of m, is of type E_0, which is subclass of C_0, the type of the body of the same method in class D.
- ”C Ok” defines the conditions that C must satisfy to be accepted as extending class D.

$$\frac{\begin{array}{c}\bar{\mathtt{x}} : \bar{\mathtt{C}}, \mathtt{this} : \mathtt{C} \vdash \mathtt{t}_0 : \mathtt{E}_0 \qquad \mathtt{E}_0 \mathrel{<:} \mathtt{C}_0 \\ \mathit{CT}(\mathtt{C}) = \mathtt{class\ C\ extends\ D\ \{\ldots\}} \\ \mathit{override}(\mathtt{m}, \mathtt{D}, \bar{\mathtt{C}} \rightarrow \mathtt{C}_0)\end{array}}{\mathtt{C}_0\ \mathtt{m}\ (\bar{\mathtt{C}}\ \bar{\mathtt{x}})\ \mathtt{\{return\ t_0;\}\ OK\ in\ C}}$$

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

Copied from reference [2]

FJ Soundness 1/2

- Given E-CastNew, a program with this term stucks.
- For the current set of FJ evaluation rules, we redefine the progress property as:

$\texttt{(A)(new Object())}$

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

Copied from reference [2]

Theorem [Progress]: Suppose $\texttt{t}$ is a closed, well-typed ~~normal form~~. Then either (1) $\texttt{t}$ is a value, or (2) $\texttt{t} \longrightarrow \texttt{t}'$ for some $\texttt{t}'$, or (3) for some evaluation context E, we can express $\texttt{t}$ as $\texttt{t} = E[\texttt{(C)(new D}(\overline{\texttt{v}}\texttt{))}]$, with $\texttt{D} \not<: \texttt{C}$.

Copied from reference [2], with dashed part.

- E[(C)(new D(v))], with !(D <: C) means the next subterm to be reduced is to stuck at E-CastNew.
  - *Solution: extend FJ with raising exceptions?*
- Proof is by industion on typing derivations: I attempted it execulding T-VAR and proving it for T-FIELD, ... etc.

FJ Soundness 2/2

Theorem [Preservation]: If $\Gamma \vdash \mathtt{t} : \mathtt{C}$ and $\mathtt{t} \longrightarrow \mathtt{t}'$, then $\Gamma \vdash \mathtt{t}' : \mathtt{C}'$ for some $\mathtt{C}' <: \mathtt{C}$.

Copied from reference [2]

- As disccused before, an apparent well-typed term may reduce to an ill-typed cast term.
- T-SCast exists so that type preservation proofs passes such failed casting.
- A type checker produces ”stupid warning” if it uses that rule.

An extension: Generic FJ

- If you are interested, lets go through pages 408 to 415 on the reference paper [3].

Optional disucssion: detecting logical type errors with FJ?

- Does FJ requires any extention to use it for detecting logical erros?
- Or, is it enough to define a class of each logical type we are interested in?
  - Is not this similar to the solution based on single-field variants in the course book [1], chapter 11, on page 139.

Suggested excercises

- Suggested excercise:
  – Should be simple: 19.4.5, 19.4.7. I alredy attempted them if any one is interested.
  – More demanding: 19.4.3, 19.4.4