九州大学学術情報リポジトリ
Kyushu University Institutional Repository

Formalization of Featherweight Java by using weak HOAS

Okumura, Kentaro
京都大学

https://hdl.handle.net/2324/1520948

出版情報：MI lecture note series. 61, pp.88-95, 2015-03-06. 九州大学マス・フォア・インダストリ研究所
バージョン：
権利関係：

KYUSHU UNIVERSITY

Formalization of Featherweight Java by using weak HOAS

*Kentaro Okumura Atsushi Igarashi

Kyoto University

Dec 4th, 2014

Featherweight Java (FJ) [Igarashi et al. 2001]

- A formal language for OO languages
- Extremely simplified Java
  - All of FJ programs are Java programs
- Some extensions
  - Featherweight GJ (FGJ) [Igarashi et al. 2001]
    - FJ+generics

An example of FJ program

```
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);
  }
}
```


About our work

- Goal
  - Formalization of FJ and FGJ, and proofs of type soundness properties of them
- Methods
  - Coq proof assistant
  - Weak Higher Order Abstract Syntax (Weak HOAS) [Despeyroux et al. 1994]
    - Methodology for formalization of variable bindings
- The Coq code is available: https://www.github.com/OKU1987/FJ_whoas

Today's talk

- Background
- About FJ
- Formalization of FJ by using weak HOAS
- Related work and conclusion

5

Featherweight Java (FJ)

- A formal language for OO languages
- Extremely simplified Java
  - All of FJ programs are Java programs
  - no assignments
- Semantics of FJ is defined by term rewriting like λ-calculus
- Pencil-and-paper proof of type soundness

6

Syntax of FJ

constructors

$$L ::= \text{class } C \text{ extends } C \ \{ \ \overline{C} \ \overline{f}; K \ \overline{M} \ \} \quad \text{(class decl.)}$$

$$M ::= C \ m(\overline{C} \ \overline{x}) \ \{ \ \text{return } e; \ \} \quad \text{(method decl.)}$$

$$e ::= x \mid e.f \mid e.m(\overline{e}) \mid \text{new } C(\overline{e}) \quad \text{(expression)}$$

including "this"

$$\overline{C} \ \overline{f}; = C_1 \ f_1; \ldots C_n \ f_n;$$

$$\overline{C} \ \overline{x} = C_1 \ x_1, \ldots, C_n \ x_n$$

$$\overline{e} = e_1, \ldots, e_n$$

7

The Pair example, again

```
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);
  }
}
```

binding

bound implicitly

8

A typing rule of FJ

$$\frac{\bar{x}:\bar{C}, \text{this}:C \vdash e_0 : E_0 \qquad E_0 <: C_0 \qquad \text{class } C \text{ extends D } \{...\} \qquad \textit{override}(m, D, \bar{C} \to C_0)}{C_0\ m(\bar{C}\ \bar{x})\ \{\ \text{return } e_0;\ \}\ \text{OK IN } C} \text{(T\_Method)}$$

Every type of FJ is a class name

A reduction rule of FJ

- A reduction rule for method invocation

$$\frac{mbody(m,C) = \bar{x}.e_0}{(\text{new } C(\bar{e})).m(\bar{d}) \longrightarrow [\bar{d}/\bar{x}, \text{new } C(\bar{e})/\text{this}]e_0} \text{(R\_Invk)}$$

"this" and parameters are replaced simultaneously

Type soundness [Igarashi et al. 2001]

If $\varnothing \vdash$ e:C and e $\longrightarrow$* e' (where e' is normal), then e' is a value v such that $\varnothing \vdash$ v:D and D<:C

Type soundness is proved by using preservation and progress theorems

Formalization of FJ

- How to represent variable bindings?
- How to represent methods?
  - Each method has different number of parameters.
- How to represent typing rules and typing contexts?
- How to represent "this"?
  - We want to
    - treat "this" as special but sometimes not
    - represent variables and "this" uniformly as much as possible

Problem of variable bindings

- Definitions of substitution and renaming are troublesome
  - How to represent fresh variables?
- There are variable bindings in FJ
  - we use weak HOAS in this work

13

Weak HOAS

- Methodology for formalizing of variable bindings
- Represent variable bindings of FJ by function abstractions of Coq
- We can utilize renaming of Coq

14

Representation of variables and expressions by using weak HOAS

- Represent variables of FJ by using type V
  - Type V is a parameter

```
Parameter V : Set.
Inductive Exp : Set :=
| e_var : V → Exp
| fd_access : Exp → F → Exp    (* F is a field name *)
| m_call : Exp → M → list Exp → Exp
                              (* M is a method name *)
| new : C → list Exp → Exp.
```

15

Representation of variable bindings by using weak HOAS

- Represent variable bindings by using function abstractions of Coq

$m(C_0\ x_0)$ { return x_0.m(); }

We ignore types of parameters for now

16

Representation of methods

m() { return new C(); } new C [] : Exp

m(C_0 x_0) { return x_0.f; }

(fun x_0 : V => ...x_0...) : V → Exp

m(C_0 x_0, C_1 x_1) { return x_1.f; }

(fun x_0 : V =>

(fun x_1 : V => ...x_1...)) : V → (V → Exp)

...

- We want to represent these methods in a single type.

Representation of method body

Inductive MB : Set :=

| mb_empty : Exp → MB (* method body *)

| mb_var : (V → MB) → MB. (* take a parameter *)

a variable binding

Examples of method body (MB type)

mb_empty (new C []) : MB

mb_var (fun x_0 : V => mb_empty (...x_0...)) : MB

mb_var (fun x_0 : V =>

mb_var (fun x_1 : V =>

mb_empty (...x_1...))) : MB

...

Representation of typing context

Regarded as binding

Represented by using an abstraction

$$\frac{\overline{x : C}, \text{this}:C \vdash e_0 : E_0 \qquad E_0 <: C_0 \qquad \text{class } C \text{ extends D } \{...\} \qquad override(m, D, C_0)}{C_0\ m(\overline{C}\ \overline{x})\ \{\ \text{return } e_0;\ \}\ \text{OK IN } C}\ (\text{T\_Method})$$

Represented by using an abstraction

Representation of typing context

- Express typing contexts by using function abstractions
  - Parameterize V, which express FJ's variables, by Ty
  - V t is the set of variables which have type t

```
Parameter V : Ty → Set.
Inductive Exp : Set :=
| e_var : forall t, V t → Exp
| fd_access : Exp → F → Exp
...
```

21

Modification of definition of method body

```
Inductive MB : Set :=
| mb_empty : Exp → MB            (* method body *)
| mb_var : (V → MB) → MB.  (* take a parameter *)
```

```
Inductive MB : Set :=
| mb_empty : Exp → MB
| mb_var : forall t : Ty, (V t → MB) → MB.
```

22

Modification of definition of method body…

- … makes MB represent types of parameters

m() { return new C(); } mb_empty (new [])

m(C_0 x_0) { return x_0.f; }
  mb_var (fun x_0 : V C_0 => mb_empty (…x_0…))

m(C_0 x_0, C_1 x_1) { return x_1.f; }
  mb_var (fun x_0 : V C_0 =>
    mb_var (fun x_1 : V C_1 => mb_empty (…x_1…)))
…

23

Representation of type judgments of expressions

```
Inductive Exp_WF : Exp → Ty → Prop :=
| T_Var : forall t (v : V t), Exp_WF (e_var v) t
| T_Fields : …
| T_Invk : …
| T_New : ….
```

24

Representation of type judgments of methods

```
Inductive Exp_WF_in_MB : MB → Ty → Prop :=
| wf_empty : forall e t, Exp_WF e t →
    Exp_WF_in_MB (mb_empty e) t
| wf_var : forall t (f : V t → MB) ty,
    (forall v : V t, Exp_WF_in_MB (f v) ty) →
    Exp_WF_in_MB (mb_var f) ty.
```

empty contexts

other contexts

$$\frac{\overline{x} : \overline{C}, \text{this}:C \vdash e_0 : E_0 \quad \ldots}{C_0\ m(\overline{C}\ \overline{x})\ \{\ \text{return}\ e_0;\ \}\ \text{OK IN}\ C}(\text{T\_Method})$$

25

Binding of this

- Every method has an implicit "parameter" of this

```
class Pair extends Object {
  Object fst;
  Object snd;
  Pair(Objest fst, Object snd) {
    super(); this.fst = fst; this.snd = snd;
  }
 Pair setfst(Object newfst) {
   return new Pair(newfst, this.snd);
 }
```

bound implicitly and exactly once

26

"this" as a usual variable

- We want to define followings uniformly:
  - substitution into arguments and "this"
  - typing rules

$$\frac{mbody(m, C) = \overline{x}.e_0}{(\text{new}\ C(\overline{e})).m(\overline{d}) \longrightarrow [\overline{d}/\overline{x}, \text{new}\ C(\overline{e})/\text{this}]e_0} \qquad \frac{(x : T) \in \Gamma}{\Gamma \vdash x : T}$$

27

Modification of MB

- to enforce implicit bindings of "this"

```
Inductive preMB : Set :=
| mb_empty : E → preMB
| mb_var : forall t : Ty, (V t → preMB) → preMB.

Inductive MB : Set :=
| mb_this : forall t : Ty, (V t → preMB) → MB.
```

"this" is bound exactly once

28

Modification of a typing rule

- following modification of MB

```
Inductive E_WF_in_MB (c:Ty) : MB → Ty → Prop :=
| wf_e_mb_this :
    forall (f : V c → preMB) ty,
    (forall v, E_WF_in_preMB c (f v) ty) →
    E_WF_in_MB c (mb_this _ f) ty.
```

$$\frac{\overline{x} : \overline{C}, \text{this}:C \vdash e_0 : E_0 \quad \ldots}{C_0\ m(\overline{C}\ \overline{x})\ \{\ \text{return}\ e_0;\ \}\ \text{OK IN}\ C}(\text{T\_Method})$$

29

Proof of type soundness

- by using former definitions on Coq

The number of lines of Coq codes.

Definition: 316 lines
Proof: 707 lines
Utilities: 81 lines

30

95

Related work

- Jinja is not Java [Klein et al. 2014]
- A Theory of Featherweight Java in Isabelle/HOL [Foster et al. 2006]
- AutoSubst [Schäfer et al. 2014]
  - Coq library for renaming and substitution
  - We can define substitution easily with some keywords
  - Using de Bruijn Index
  - Mutually recursive types are not supported

31

Conclusion

- We formalize a calculus for OO languages by using higher order abstract syntax
- We show that we can apply a HOAS method even if each method of FJ has different number of parameters
- We treat problems of “this”
- The Coq code of our formalization is available: https://www.github.com/OKU1987/FJ_whoas
- On going work: Formalization of FGJ

32