Logical Relations for a Manifest Contract Calculus

Taro Sekiyama Atsushi Igarashi

Kyoto University

Manifest Contract Calculus [1]

- A typed lambda calculus with (higher-order) *software contracts*
- *hybrid* checking of software contracts
  - Static type system: refinement type $\{x{:}T \mid e\}$
    e.g. $\{x{:}\mathsf{int} \mid 0 < x\}$
  - Dynamic checking: cast $\langle T_1 \Rightarrow T_2 \rangle^{\ell}$
    e.g. $\langle \mathsf{int} \Rightarrow \{x{:}\mathsf{int} \mid x < 0\} \rangle^{\ell}$

[1] Knowles and Flanagan, 2010

Programming in Manifest Contract Calculus

$$\mathrm{div} : \mathrm{int} \rightarrow \{x{:}\mathrm{int} \mid 0 \neq x\} \rightarrow \mathrm{int}$$

div "abc" 2 (* Compiler error *)

div 6 0 (* Compiler error *)

(* Compiler doesn't know that y is non-zero *)
(fun y : int. div 6 y)

Programming in Manifest Contract Calculus

$\text{div} : \text{int} \rightarrow \{x{:}\text{int} \mid 0 \neq x\} \rightarrow \text{int}$

div “abc” 2 (* Compiler error *)

div 6 0 (* Compiler error *)

(* Compiler inserts a cast *)

$(\texttt{fun}\ y : \text{int}.\ \text{div}\ 6\ (\langle \text{int} \Rightarrow \{x{:}\text{int} \mid 0 \neq x\}\rangle^{\ell}\ y))$

Previous Work: Upcast Elimination

Upcast Elimination [1,2]

An upcast and an identity function are contextually equivalent

An upcast is a cast from a type to its supertype

- $\langle \{x{:}\mathsf{int} \mid 0 < x\} \Rightarrow \mathsf{int} \rangle^{\ell}$
- $\langle \{x{:}\mathsf{int} \mid \mathsf{is\_square}\ x\} \Rightarrow \{x{:}\mathsf{int} \mid 0 < x\} \rangle^{\ell}$

Upcast elimination is useful for optimization

[1] Knowles and Flanagan, 2010
[2] Belo et al., 2011

Previous Work: Correctness of Proofs

Previous work

- tried to prove upcast elimination by using *logical relations*
- didn't really prove soundness of the logical relations w.r.t contextual equivalence

	$\lambda_{\mathrm{H}}^{[1]}$	$\mathrm{F_H}^{[2]}$
$\langle T_1 \Rightarrow T_2 \rangle^\ell \simeq \texttt{fun x.x}$	proved	proved
$\simeq \subseteq \approx$	flawed	not proved
$\langle T_1 \Rightarrow T_2 \rangle^\ell \approx \texttt{fun x.x}$	not proved	not proved

$\approx$: contextual equivalence $\simeq$: logical relation

[1] Knowles and Flanagan, 2010 [2] Belo et al., 2011

Logical Relations for a Manifest Contract Calculus, Fixed

Taro Sekiyama Atsushi Igarashi

Kyoto University

This Work

This work

- *fixes* the flaws of previous work
- introduces $\mathrm{F}_{\mathrm{H}}^{\mathtt{fix}}$
  - a polymorphic manifest contract calculus with *fixed*-point operator
  - non-termination is only *effect* in $\mathrm{F}_{\mathrm{H}}^{\mathtt{fix}}$

	λ_{H}	F_{H}	$\mathrm{F}_{\mathrm{H}}^{\mathtt{fix}}$
Subsumption rule	✓	×	×
Polymorphic types	×	✓	✓
Fixed-point operator	×	×	✓

Contribution

- *Semi-typed* contextual equivalence
- A sound logical relation w.r.t *semi-typed* contextual equivalence
- Proof of upcast elimination by using the logical relation above
  - We *believe* correctness of our proof :-)

	$\lambda_{\mathtt{H}}$	$\mathrm{F}_{\mathtt{H}}$	$\mathrm{F}_{\mathtt{H}}^{\mathtt{fix}}$
$\langle T_1 \Rightarrow T_2 \rangle^{\ell} \simeq \mathtt{fun\ x.x}$	proved	proved	proved
$\simeq \subseteq \approx$	flawed	not proved	proved
$\langle T_1 \Rightarrow T_2 \rangle^{\ell} \approx \mathtt{fun\ x.x}$	not proved	not proved	proved

Contents

Contents

Overview of $\mathrm{F}_{\mathtt{H}}^{\mathtt{fix}}$

$\mathrm{F}_{\mathtt{H}}^{\mathtt{fix}}$ is a typed lambda calculus with

- polymorphic types,
- refinement types $\{x{:}T \mid e\}$,
- dependent function types $x{:}T_1 \rightarrow T_2$,
- casts $\langle T_1 \Rightarrow T_2 \rangle^{\ell}$, and
- fixed-point operator (recursive functions)

	$\lambda_{\mathtt{H}}$	$\mathrm{F}_{\mathtt{H}}$	$\mathrm{F}_{\mathtt{H}}^{\mathtt{fix}}$
Subsumption rule	✓	×	×
Polymorphic types	×	✓	✓
Recursive functions	×	×	✓

Types

Refinement types: $\{x{:}T \mid e\}$

- denote a set of values which
  - are in T
  - satisfy the contract (boolean expression) e
- e.g. $\{x{:}\mathrm{int} \mid 0 < x\} = \{1, 2, 3, \ldots\}$

Dependent function types: $x{:}T_1 \rightarrow T_2$

- denote a set of functions which
  - accept values v of T_1
  - return values of $T_2\,[v/x]$
- e.g. $x{:}\mathrm{int} \rightarrow \{y{:}\mathrm{int} \mid x < y\}$

Dynamic Checking: Cast

Casts: $\langle T_1 \Rightarrow T_2 \rangle^\ell$

- accept values v of T_1
- check whether v can behave as T_2
  - If the checking fails, the cast is blamed with label ℓ
- e.g. $\langle \mathsf{int} \Rightarrow \{x{:}\mathsf{int} \mid 0 < x\} \rangle^\ell$

$$\langle \mathsf{int} \Rightarrow \{x{:}\mathsf{int} \mid 0 < x\} \rangle^\ell\ 0 \rightsquigarrow^* \Uparrow\ell$$
$$\langle \mathsf{int} \Rightarrow \{x{:}\mathsf{int} \mid 0 < x\} \rangle^\ell\ 2 \rightsquigarrow^* 2$$

Digression: Pitfall of A-Normal Form

- At first, we gave A-normal form as syntax
  - following [3] which uses A-normal form to simplify the definition and the proof
  - $e ::= v_1\ v_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \mid \cdots$
- It is difficult to prove even *type soundness*
  - to require substitution of *terms*
  - A-normal form is *not* closed under substitution of terms

$$\frac{\Gamma \vdash e_1 : T_1 \quad \Gamma, x{:}T_1 \vdash e_2 : T_2}{\Gamma \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : T_2\,[e_1/x]}$$

[3] Pitts, 2005

Contents

Review: (Typed) Contextual Equivalence

$e_1 \approx_{typed} e_2 : T$

- e_1 and e_2 have the same observable result under any contexts
  - which are well-typed and accept any terms of T
- e_1 and e_2 are typed at the same type T

$$(\mathtt{fun}\ x : \mathsf{int}.\ 0) \approx_{typed} (\mathtt{fun}\ x : \mathsf{int}.\ x * 0) : \mathsf{int} \rightarrow \mathsf{int}$$
$$(\mathtt{fun}\ x : \mathsf{int}.\ 0) \not\approx_{typed} (\mathtt{fun}\ x : \mathsf{int}.\ x + 2) : \mathsf{int} \rightarrow \mathsf{int}$$

$$(\mathtt{fun}\ x : \mathsf{int}.\ 0) \not\approx_{typed} (\mathtt{fun}\ x : \mathsf{bool}.\ 0) : \mathsf{int} \rightarrow \mathsf{int}$$

Problem

- Upcast elimination doesn't hold in typed contextual equivalence
  - An upcast and an identity function may have different types
  - Note lack of a subsumption rule

$$\frac{\langle T_1 \Rightarrow T_2 \rangle^{\ell}}{T_1 \to T_2} \quad \Bigg| \quad \frac{\texttt{fun}\ x : T_1.\ x}{T_1 \to T_1} \quad \Bigg| \quad \frac{\texttt{fun}\ x : T_2.\ x}{T_2 \to T_2}$$

- We must relax typed contextual equivalence

Semi-Typed Contextual Equivalence

$e_1 \approx e_2 : T$

- e_1 and e_2 have the same observable result under any well-typed contexts
- Only e_1 is typed at T
  - e_2 can even be ill-typed

$$(\texttt{fun}\ x : \mathsf{int}.\ 0) \approx (\texttt{fun}\ x : \mathsf{int}.\ x * 0) : \mathsf{int} \rightarrow \mathsf{int}$$
$$(\texttt{fun}\ x : \mathsf{int}.\ 0) \not\approx (\texttt{fun}\ x : \mathsf{int}.\ x + 2) : \mathsf{int} \rightarrow \mathsf{int}$$

$$(\texttt{fun}\ x : \mathsf{int}.\ 0) \approx (\texttt{fun}\ x : \mathsf{bool}.\ 0) : \mathsf{int} \rightarrow \mathsf{int}$$

Formal Definition

Definition

Semi-typed contextual equivalence $\approx$ is the largest set satisfying the following:

1. If $\Gamma \vdash e_1 \approx e_2 : T$, then $\Gamma \vdash e_1 : T$
2. If $\emptyset \vdash e_1 \approx e_2 : T$, then e_1 and e_2 have the same observable result
3. Reflexivity, Transitivity, (Typed) Symmetry
4. Compatibility
5. Substitutivity

Compatibility and Substitutivity Rules

Choose *typed* terms for substitution on types

- so that the type after the substitution is well-formed

E.g.

Compatibility: term application

$$\frac{\Gamma \vdash e_{11} \approx e_{21} : (x{:}T_1 \to T_2) \quad \Gamma \vdash e_{12} \approx e_{22} : T_1}{\Gamma \vdash e_{11}\ e_{12} \approx e_{21}\ e_{22} : T_2\,[e_{12}/x]}$$

Substitutivity: value substitution

$$\frac{\Gamma, x{:}T_1, \Gamma' \vdash e_1 \approx e_2 : T_2 \quad \Gamma \vdash v_1 \approx v_2 : T_1}{\Gamma, \Gamma'[v_1/x] \vdash e_1\,[v_1/x] \approx e_2\,[v_2/x] : T_2\,[v_1/x]}$$

Contents

Overview of Logical Relation

$e_1 \simeq e_2 : T$

- $\simeq$ is defined by using
  - basic ideas of the logical relation for $\mathrm{F_H}$[2]
  - $\top\top$-closure[3]
    - A method to give a logical relation to a lambda calculus with recursive functions
- Only e_1 is typed
  - similarly to semi-typed contextual equivalence

[2] Belo et al., 2011
[3] Pitts, 2005

How to Define Logical Relation by ⊤⊤

1. Define value relations for base types

bool: {(true,true), (false,false)}

int: {...,(-1,-1),(0,0),(1,1),...}

How to Define Logical Relation by $\top\top$

1. Define value relations for base types
2. Define term relations for base types by operation $\top\top$
   - $\top\top$ expands value relations to term relations

bool : {(true, not false),(true && true, true) ...}

int: $\{(1+1,2),(0*3,0+0),\ldots\}$

$$\boxed{\text{Value relation}} \xrightarrow{\boxed{\top\top}} \boxed{\text{Term relation}}$$

How to Define Logical Relation by $\top\top$

1. Define value relations for base types
2. Define term relations for base types by operation $\top\top$
3. Define value relations for complex types

$$\mathsf{int} \rightarrow \mathsf{int} : \{(\mathsf{succ}, \mathtt{fun}\ \mathtt{x}.\mathtt{x} + 1), \ldots\}$$

$$\begin{tikzcd} \boxed{\text{Value relation}} \arrow[r, "\boxed{\top\top}"] & \boxed{\text{Term relation}} \arrow[l, bend left] \end{tikzcd}$$

How to Define Logical Relation by $\top\top$

1. Define value relations for base types
2. Define term relations for base types by operation $\top\top$
3. Define value relations for complex types
4. Define term relations for complex types by operation $\top\top$

$$
\begin{tikzcd}
\boxed{\text{Value relation}} \arrow[r, "\boxed{\top\top}"] & \boxed{\text{Term relation}} \arrow[l, bend left]
\end{tikzcd}
$$

How to Define Logical Relation by $\top\top$

1. Define value relations for base types
2. Define term relations for base types by operation $\top\top$
3. Define value relations for complex types
4. Define term relations for complex types by operation $\top\top$

$$\vdots$$

$$\begin{tikzcd}
\boxed{\text{Value relation}} \arrow[r, "\boxed{\top\top}"] & \boxed{\text{Term relation}} \arrow[l, bend left]
\end{tikzcd}$$

Relations for Closed Terms

- Value relation: $T(\theta, \delta)^{\mathsf{val}}$
- Term relation: $T(\theta, \delta)^{\mathsf{tm}}$

Here,

- θ is a valuation for type variables in T
  - $\theta = \{\alpha \mapsto (r, T_1, T_2), ...\}$

    r is a term relation and an interpretation of α
  - Notation: $\theta_i = \{(\alpha \mapsto T_i), ...\}$
- δ is a valuation for variables in T
  - $\delta = \{x \mapsto (v_1, v_2), ...\}$
  - Notation: $\delta_i = \{(x \mapsto v_i), ...\}$

Value/Term Relation: Base Types

Base type: B

Value Relation

$(v_1, v_2) \in B(\theta, \delta)^{\mathsf{val}}$ iff
$v_1 = v_2$ and v_1 is a constant of B

Term Relation

$B(\theta, \delta)^{\mathsf{tm}} = (B(\theta, \delta)^{\mathsf{val}})^{\top\top}$

Value/Term Relation: Dependent Function Types

Value Relation

$(v_1, v_2) \in (x{:}T_1 \to T_2)(\theta, \delta)^{\mathsf{val}}$ iff
for any $(v_1', v_2') \in T_1(\theta, \delta)^{\mathsf{tm}}$,
$\quad (v_1\ v_1', v_2\ v_2') \in T_2(\theta, \delta\{x \mapsto v_1', v_2'\})^{\mathsf{tm}}$

Term Relation

$(x{:}T_1 \to T_2)(\theta, \delta)^{\mathsf{tm}} = ((x{:}T_1 \to T_2)(\theta, \delta)^{\mathsf{val}})^{\top\top}$

Value/Term Relation: Refinement Types

Value Relation

$(v_1, v_2) \in \{x{:}T \mid e\}(\theta, \delta)^{\mathsf{val}}$ iff

- $(v_1, v_2) \in T(\theta, \delta)^{\mathsf{tm}}$
- $\theta_1(\delta_1(e\,[v_1/x])) \leadsto^* \mathsf{true}$
- $\theta_2(\delta_2(e\,[v_2/x])) \leadsto^* \mathsf{true}$

Term Relation

$\{x{:}T \mid e\}(\theta, \delta)^{\mathsf{tm}} = (\{x{:}T \mid e\}(\theta, \delta)^{\mathsf{val}})^{\top\top}$

Logical Relation for Open Terms

Definition (Logical Relation for Open Terms)

$\Gamma \vdash e_1 \simeq e_2 : T$ *iff*

1. $\Gamma \vdash e_1 : T$
2. $(\theta_1(\delta_1(e_1)), \theta_2(\delta_2(e_2))) \in T(\theta, \delta)^{\mathsf{tm}}$
   where $\Gamma \vdash \theta; \delta$

- e_1 and e_2 are related for well-formed substitution θ and δ

Properties of Logical Relation

Theorem (Soundness)

If $\Gamma \vdash e_1 \simeq e_2 : T$, *then* $\Gamma \vdash e_1 \approx e_2 : T$

- Prove that $\simeq$ satisfies the properties defining $\approx$

Theorem (Completeness w.r.t Typed Terms)

If $\Gamma \vdash e_1 \approx e_2 : T$ *and* $\Gamma \vdash e_2 : T$,
then $\Gamma \vdash e_1 \simeq e_2 : T$

- An orthodox method doesn't go through

Soundness: Overview of Proof

We must prove that for soundness

the logical relation satisfies

- reflexivity, transitivity, typed symmetry
- compatibility
- substitutivity

Note that

- it suffices to prove only compatibility and substitutivity in [3]
- all the properties are proved in this work

[3] Pitts, 2005

Contents

Upcast Elimination

Upcast Elimination

An upcast and an identity function are contextually equivalent

Lemma

If $\Gamma \vdash T_1 <: T_2$, then
$\Gamma \vdash \langle T_1 \Rightarrow T_2 \rangle^{\ell} \simeq (\texttt{fun}\ x : T_1.\ x) : T_1 \rightarrow T_2$

Corollary

If $\Gamma \vdash T_1 <: T_2$, then
$\Gamma \vdash \langle T_1 \Rightarrow T_2 \rangle^{\ell} \approx (\texttt{fun}\ x : T_1.\ x) : T_1 \rightarrow T_2$

Contents

Conclusion

- A sound logical relation w.r.t semi-typed contextual equivalence
- Proof of upcast elimination

Technically,

- $\top\top$-closure works in manifest contract calculus with non-termination
  - The proofs of the properties are troublesome
- "Semi-typedness" doesn't complicate the proof of soundness
  - affects the proof of completeness

Future Work

- Unrestricted completeness
  - removal of "typedness" assumption
- Correctness of other optimizations
- Effects other than non-termination