A Bisimulation for Dynamic Sealing

Eijiro Sumii
Benjamin C. Pierce
University of Pennsylvania

Modularity by Abstraction, Abstraction by Typing

- **Modularity** is crucial for managing large systems
- **Abstraction** is the primary method of achieving modularity
- **Type abstraction** is a common way of enforcing abstraction in programming languages
  - Almost as old [Liskov 73] as structured programming [Dijkstra 68]

A Classic Example: Complex Numbers

```
interface Complex
  abstype t
  make_complex : real × real → t
  get_re : t → real
  get_im : t → real
  multiply : t × t → t
end
```

Cartesian Implementation

module CartesianComplex implements Complex

abstype t = real × real

make_complex(x, y) = (x, y)

get_re(x, y) = x

get_im(x, y) = y

multiply((x$_1$, y$_1$), (x$_2$, y$_2$)) =

$(x_1 \times x_2 - y_1 \times y_2, x_1 \times y_2 + y_1 \times x_2)$

end

Polar Implementation

module PolarComplex implements Complex

abstype t = real $\times$ real

make_complex(x, y) =

(sqrt(x $\times$ x + y $\times$ y), atan2(y, x))

get_re(r, θ) = r $\times$ cos(θ)

get_im(r, θ) = r $\times$ sin(θ)

multiply((r_1, θ_1), (r_2, θ_2)) = ($r_1 \times r_2$, $\theta_1 + \theta_2$)

end

Abstraction as Equivalence

- Abstraction is indeed achieved iff the two implementations are **contextually equivalent** (i.e., cannot be distinguished by their users)

  CartesianComplex $\equiv$ PolarComplex : Complex

  $\Updownarrow$

  For any C : Complex $\rightarrow$ unit, C[CartesianComplex] terminates iff C[PolarComplex] does

  - Can be proved via **logical relations**
  - In this talk, only convergence/divergence is observed (not timing, power consumption, etc.)

Problem

- Type abstraction doesn't work in today's open untyped program environments
  - Abstraction is lost if data is written into a file or sent over the network, where not all programs are statically typed
    - You cannot "type-check the Internet"

Pseudo-example:

```
send(PolarComplex.make_complex(1.0, 2.0)) |
CartesianComplex.get_re(receive())
```

A Solution

- Use a more dynamic method of information hiding: **sealing** (≈ perfect encryption)
  - As old as type abstraction [Morris 73]
  - Interest renewed [Pierce-Sumii 2000, Leifer-Peskine-Sewell 2003, Rossberg 2003, etc.]

Abstraction by Sealing

- A fresh, secret **seal** (or key) is generated for each abstract type
- Abstract data is **sealed** (or "encrypted") when going out of a module, and **unsealed** (or "decrypted") when coming back

Illegal access causes failure of unsealing (which prevents failure of abstraction)

- No need to type-check the Internet

CartesianComplex with Sealing

module CartesianComplex

make_complex(x, y) = $\{(x, y)\}_k$

get_re(c) = let $\{(x, y)\}_k$ = c in x

get_im(c) = let $\{(x, y)\}_k$ = c in y

multiply(c_1, c_2) =

let $\{(x_1, y_1)\}_k = c_1$ in

let $\{(x_2, y_2)\}_k = c_2$ in

$\{(x_1 \times x_2 - y_1 \times y_2, x_1 \times y_2 + y_1 \times x_2)\}_k$

end

PolarComplex with Sealing

module PolarComplex

make_complex(x, y) =

$\{(\mathrm{sqrt}(x \times x + y \times y), \mathrm{atan2}(y, x))\}_{k'}$

get_re(c) = let $\{(r, \theta)\}_{k'}$ = c in $r \times \cos(\theta)$

get_im(c) = let $\{(r, \theta)\}_{k'}$ = c in $r \times \sin(\theta)$

multiply(c_1, c_2) =

let $\{(r_1, \theta_1)\}_{k'} = c_1$ in

let $\{(r_2, \theta_2)\}_{k'} = c_2$ in

$\{(r_1 \times r_2, \theta_1 + \theta_2)\}_{k'}$

end

Question

- Is this use of sealing correct? That is, does it indeed achieve abstraction?

Sub-questions:

- How to <u>state</u> the abstraction property?
  - Standard definition of contextual equivalence needs to be generalized (taking "knowledge of the environment about seals" into account)
- How to <u>prove</u> it?
  - Logical relations rely on types and cannot be applied in untyped setting

Main Results of This Work

- Definition of λ_{seal}, untyped call-by-value λ-calculus extended with sealing
- Generalization of contextual equivalence for λ_{seal}
- Development of a <u>sound</u> and <u>complete</u> bisimulation proof technique for λ_{seal}

  Examples:
  - Data abstraction (complex numbers)
  - Protocol encoding (Needham-Schroeder-Lowe)

A Frequently Asked Question

- What are different from bisimulations for the spi-calculus? [Abadi-Gordon 98, Boreale-Nicola-Pugliese 99, Borgstroem-Nestmann 02, etc.]
  - (Higher-order) functions
    - Hard to encode into spi-calculus without losing abstraction (against untyped environment, in particular)
  - Non-trivial generalization of contextual equivalence
  - Simpler definition of bisimulation
    - No complications like "frame", "theory", "analysis" or "synthesis"

Outline

- Introduction
  - Abstraction by typing
  - Abstraction by sealing
- Syntax and semantics of λ_{seal}
- Contextual equivalence in λ_{seal}
- Bisimulation for λ_{seal}
- Related work and conclusions

Outline

- Introduction
  - Abstraction by typing
  - Abstraction by sealing
- Syntax and semantics of λ_{seal}
- Contextual equivalence in λ_{seal}
- Bisimulation for λ_{seal}
- Related work and conclusions

Syntax of λ_{seal}

Standard untyped call-by-value λ-calculus extended with primitives for sealing

- Seal: $k \in K$ (countably infinite set of seals)
- Fresh seal generation: $\nu x.\ e$
- Sealing: $\{e_1\}_{e2}$
- Unsealing: $\text{let } \{x\}_{e1} = e_2 \text{ in } e_3 \text{ else } e_4$
  - $\text{let } \{x\}_k = \{v\}_k \text{ in } e_3 \text{ else } e_4 \ \rightarrow\ [v/x]e_3$
  - $\text{let } \{x\}_k = \{v\}_{k'} \text{ in } e_3 \text{ else } e_4 \ \rightarrow\ e_4$ (if $k \neq k'$

Semantics of λ_{seal}

- Big-step **evaluation**

$$\langle s \rangle\ e \Downarrow \langle t \rangle\ v$$

where s and t are **seal sets** before and after the evaluation

  - E.g.

$$\frac{k \notin s \qquad \langle s \cup \{k\} \rangle\ [k/x]e \Downarrow \langle t \rangle\ v}{\langle s \rangle\ \nu x.\ e \Downarrow \langle t \rangle\ v}\ \text{(E-New)}$$

Outline

- Introduction
  - Abstraction by typing
  - Abstraction by sealing
- Syntax and semantics of λ_{seal}
- Contextual equivalence in λ_{seal}
- Bisimulation for λ_{seal}
- Related work
- Conclusions

Contextual Equivalence: Problem

- Standard definition doesn't suffice, e.g.,

$$\lambda c.\ \text{let } \{(x, y)\}_k = c \text{ in } x \text{ else } \bot$$

$$\equiv_?$$

$$\lambda c.\ \text{let } \{(r, \theta)\}_{k'} = c \text{ in } r \times \cos(\theta) \text{ else } \bot$$

- The answer depends on **knowledge of the context**:
  - If it knows any $\{(x, y)\}_k$ and $\{(r, \theta)\}_{k'}$ such that $x \neq r \times \cos(\theta)$, then no.
  - If it knows no such values, then yes.

Contextual Equivalence: Solution

- A binary relation R over values is ***abstractive*** if:

  For any $(v_1, v'_1), \ldots, (v_n, v'_n) \in R$,
  for any seal-free term e,
  $[v_1,\ldots,v_n/x_1,\ldots,x_n]e$ terminates iff
  $[v'_1,\ldots,v'_n/x_1,\ldots,x_n]e$ does.

- Contextual equivalence $\equiv$ is the set of all such R's
  - Intuition: R represents environment's knowledge
    - For simplicity, we consider closed values only
    - Strictly speaking, each $R \in \equiv$ is annotated with seal sets s and s'

Contextual Equivalence: Examples

- $\{(\{(1, 1)\}_k, \{(\sqrt{2}, \pi/4)\}_{k'}),$
  $(\lambda c.\ \text{let } \{(x, y)\}_k = c \text{ in } x \text{ else } \bot,$
  $\lambda c.\ \text{let } \{(r, \theta)\}_{k'} = c \text{ in } r \times \cos(\theta) \text{ else } \bot)\} \in \equiv$
- $\{(\{(\sqrt{2}, \pi/4)\}_k, \{(1, 1)\}_{k'}),$
  $(\lambda c.\ \text{let } \{(x, y)\}_k = c \text{ in } x \text{ else } \bot,$
  $\lambda c.\ \text{let } \{(r, \theta)\}_{k'} = c \text{ in } r \times \cos(\theta) \text{ else } \bot)\} \notin \equiv$
- $\{(\{(\sqrt{2}, \pi/4)\}_k, \{(1, 1)\}_{k'})\} \in \equiv$
- $\{(\lambda c.\ \text{let } \{(x, y)\}_k = c \text{ in } x \text{ else } \bot,$
  $\lambda c.\ \text{let } \{(r, \theta)\}_{k'} = c \text{ in } r \times \cos(\theta) \text{ else } \bot)\} \in \equiv$
- $\{(\{(1, 1)\}_k, \{(\sqrt{2}, \pi/4)\}_{k'}), (k, k')\} \notin \equiv$

Outline

- Introduction
  - Abstraction by typing
  - Abstraction by sealing
- Syntax and semantics of λ_{seal}
- Contextual equivalence in λ_{seal}
- Bisimulation for λ_{seal}
- Related work
- Conclusions

Bisimulation: Motivation

- In general, contextual equivalence is hard to prove <u>directly</u> $\Rightarrow$ proof technique necessary
  - Logical relations are not applicable since our setting is untyped
  - We consider **bisimulation** (cf. applicative bisimulation [Abramsky 90])

Bisimulation: Definition (1/3)

Intuition: each condition on a bisimulation excludes pairs of values distinguishable by the environment

A bisimulation X is a set of binary relations over values such that, for every $R \in X$,

- For each $(v, v') \in R$, v and v' are values of the same kind (i.e., both are constants, functions, tuples, seals, or sealed values)
- For each $(c, c') \in R$, we have $c = c'$
- For each $((v_1, \ldots, v_n), (v'_1, \ldots, v'_{n'})) \in R$, we have $n = n'$ and $R \cup \{(v_i, v'_i)\} \in X$ for each i

Bisimulation: Definition (2/3)

- For each $(k_1, k'_1) \in R$ and $(k_2, k'_2) \in R$, we have $k_1 = k_2 \Leftrightarrow k'_1 = k'_2$
  - Rationale: The context can test (only) the equality of two seals, via sealing under one seal and unsealing under the other seal
- For each $(\{v\}_k, \{v'\}_{k'}) \in R$, we have either:
  - $(k, k') \in R$ and $R \cup \{(v, v')\} \in X$, or else
  - $(k, k'') \notin R$ and $(k'', k') \notin R$ for any k''
    - Rationale: Either the context knows both seals and can unseal both sealed values, or it knows none of the seals.

Bisimulation: Definition (3/3)

- For each $(\lambda x.\ e, \lambda x.\ e') \in R$,
  1. Take any fresh $(k_1, k'_1), \ldots, (k_m, k'_m)$ and let $S = R \cup \{(k_1, k'_1), \ldots, (k_m, k'_m)\}$
  2. Take any $(u_1, u'_1), \ldots, (u_n, u'_n) \in S$ and any seal-free term d with free variables $x_1, \ldots, x_n$
  3. Let $v = [u_1,\ldots,u_n/x_1,\ldots,x_n]d$ and $v' = [u'_1,\ldots,u'_n/x_1,\ldots,x_n]d$
  4. Apply $\lambda x.\ e$ to v and $\lambda x.\ e'$ to v'
  5. Then, both converge or both diverge
  6. If they converge, let the results be w and w'
  7. Then, $S \cup \{(w, w')\} \in X$

Bisimulation: Example

The following set (of binary relations over values) is a bisimulation:

{{(CartesianComplex, PolarComplex),
(CartesianComplex.make_complex,
PolarComplex.make_complex),
(CartesianComplex.get_re, PolarComplex.get_re),
(CartesianComplex.get_im, PolarComplex.get_im),
(CartesianComplex.multiply, PolarComplex.multiply)} $\cup$
$\{(x, x) \mid x : \text{real}\} \cup$
$\{(\{(x, y)\}_k, \{(r, \theta)\}_{k'}) \mid x = r \cos \theta \wedge y = r \sin \theta \} \cup$
$\{(k_1, k_1'), \ldots, (k_n, k_n')\} \mid$
$k \notin \{k_1, \ldots, k_n\} \wedge k' \notin \{k'_1, \ldots, k'_n\}\}$

Bisimulation: Properties

Lemmas:

1. Contextual equivalence is a bisimulation
   - Proof: By checking the conditions of bisimulation
2. Bisimilar values put in any seal-free context are observationally equivalent <u>and such forms are preserved by evaluation</u>
   - Proof: By induction on the derivation of evaluation

Theorem [soundness & completeness]:
Bisimilarity (the largest bisimulation) coincides with contextual equivalence

Another Example: Encoding Security Protocols

- Shows the power of λ_{seal} and its bisimulation

Ideas:

- Encryption is encoded as sealing
- Protocol is encoded as a tuple of public keys and principals
- Senders are encoded as the values being sent
- Receivers are encoded as functions
- Contexts play the role of attackers, network and scheduler by applying receivers to senders

Related Work

- Bisimulations for the spi-calculus
- Logical relations for encryption [Sumii-Pierce 2001]
  - Cannot be used in untyped settings
  - Even in typed settings, do not scale well for richer languages with recursive functions/types etc.
- Applicative bisimulation [Abramsky 90]
  - Soundness proof is <u>very</u> hard [Howe 96]
  - Our soundness proof is much easier thanks to variations in arguments of functions

Conclusions

- Summary: We defined λ_{seal} and developed a <u>sound</u> and <u>complete</u> proof technique for "type abstraction without types"
- Future work:
  - Full abstraction for general, type-directed translation of type abstraction [Pierce-Sumii 00]
  - Similar bisimulations for other forms of information hiding (such as information flow/access control and type abstraction)