

# Environmental Bisimulations for Program Equivalences

Eijiro Sumii  
(Tohoku University)



In collaboration with:  
Benjamin C. Pierce, Davide Sangiorgi,  
Naoki Kobayashi, and Nobuyuki Sato

# FLOPS 2010 : FrontPage

<http://www.kb.ecei.tohoku.ac.jp/flops2010/wiki/index.php?FrontPage>

---

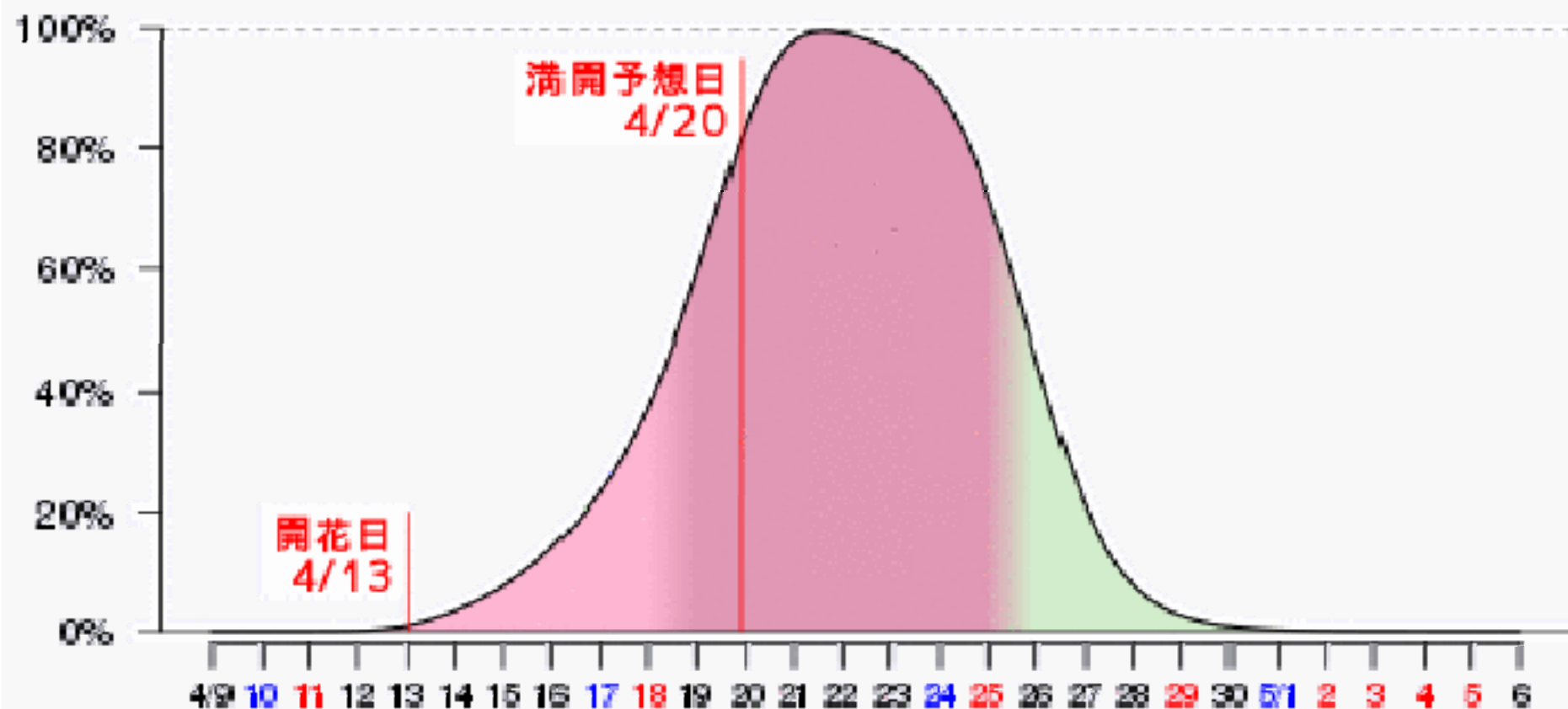
FLOPS 2010  
Tenth International Symposium on Functional and Logic Programming  
April 19-21, 2010  
[Sendai](#), [Japan](#)

Shortcuts: [Registration](#), [Program](#), [Venue](#), [Excursion](#), [More Excursions](#)



2010-04-14

# 仙台



見頃：4/19～4/25ころ



# FLOPS 2010 : Excursion

<http://www.kb.ecei.tohoku.ac.jp/flops2010/wiki/index.php?Excursion>

---

## Excursion †

Half-day trip to [Hiraizumi](#) 📍 is planned on the afternoon of the second day (April 20, Tuesday).





# Environmental Bisimulations for Program Equivalences

Eijiro Sumii  
(Tohoku University)

In collaboration with:  
Benjamin C. Pierce, Davide Sangiorgi,  
Naoki Kobayashi, Nobuyuki Sato

# What are environmental bisimulations?

A theory for proving equivalences of programs in various languages (higher-order, in particular)

- Devised for  $\lambda$ -calculus with encryption [Sumii-Pierce POPL04]
- Adapted for polymorphic  $\lambda$ -calculus [Sumii-Pierce POPL05], untyped  $\lambda$ -calculus with references [Koutavas-Wand POPL06] [Sangiorgi-Kobayashi-Sumii LICS07] and deallocation [Sumii ESOP09], etc.

# This talk

Two instances of environmental bisimulations, for

- I. Polymorphic  $\lambda$ -calculus with references [Sumii CSL09], and
- II. Higher-order  $\pi$ -calculus (concurrent language with message passing) with encryption [Sato-Sumii APLAS09] (if time allows).

# Part I: Environmental Bisimulations for Polymorphic $\lambda$ -calculus with References



# Executive Summary

Sound and complete "proof method"  
for contextual equivalence  
in a language with

- Higher-order functions,
- First-class references (like ML), and
- Abstract data types

**Caveat: the method is not fully automatic!**

- The equivalence is (of course) undecidable in general
- Still, it successfully proved all known examples

# (Very) General Motivation

1. Equations are important in science
  - $1 + 2 = 3$ ,  $x + y = y + x$ ,  $E = mc^2$ , ...
2. Computing is (should be) a science
3. Therefore, equations are important in (so-called) computer science
4. Computing is described by programs
5. Therefore, equivalence of programs is important!

# Program Equivalence as Contextual Equivalence

In general, equations should be preserved under any context

- E.g.,  $x + y = y + x$  implies  $(x + y) + z = (y + x) + z$  by considering the context  $[ ] + z$

⇒ Contextual equivalence:

Two programs "give the same result" under any context

- Termination/divergence suffices for the "result"

# Contextual Equivalence: Definition

Two programs  $P$  and  $Q$  are contextually equivalent if, for any context  $C$ ,

$C[P]$  terminates  $\Leftrightarrow C[Q]$  terminates

- $C[P]$  (resp.  $C[Q]$ ) means "filling in" the "hole"  $[ ]$  of  $C$  with  $P$  (resp.  $Q$ )

# Example: Two Implementations of Mutable Integer Lists

```
(* pseudo-code in  
  imaginary ML-like language *)  
signature S  
  type t (* abstract *)  
  val nil : t  
  val cons : int → t → t  
  val setcar : t → int → unit  
  (* car, cdr, setcdr, etc. *)  
end
```

# First Implementation

structure L

type t =

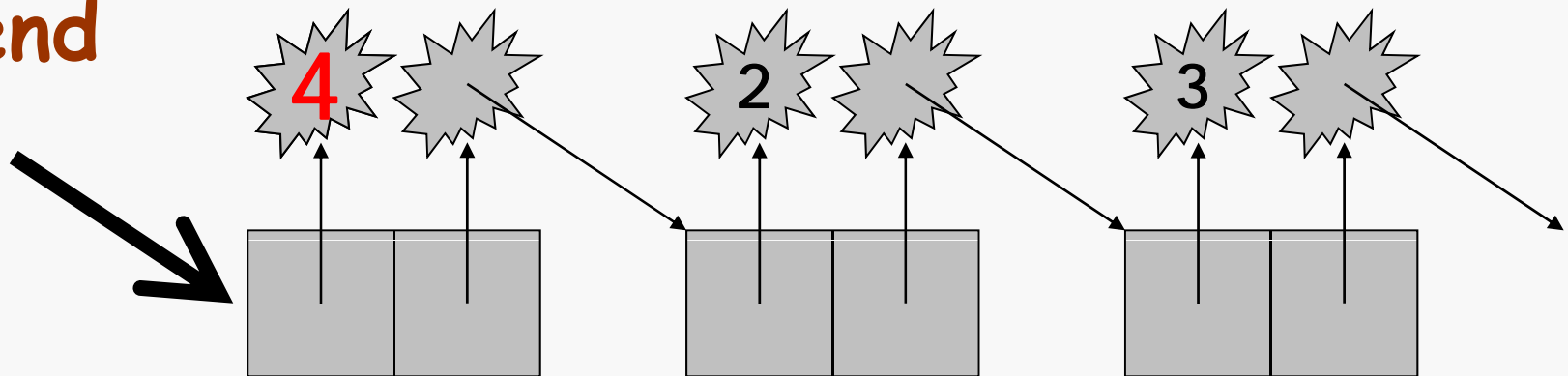
Nil | Cons of (int ref \* t ref)

let nil = Nil

let cons a d = Cons(ref a, ref d)

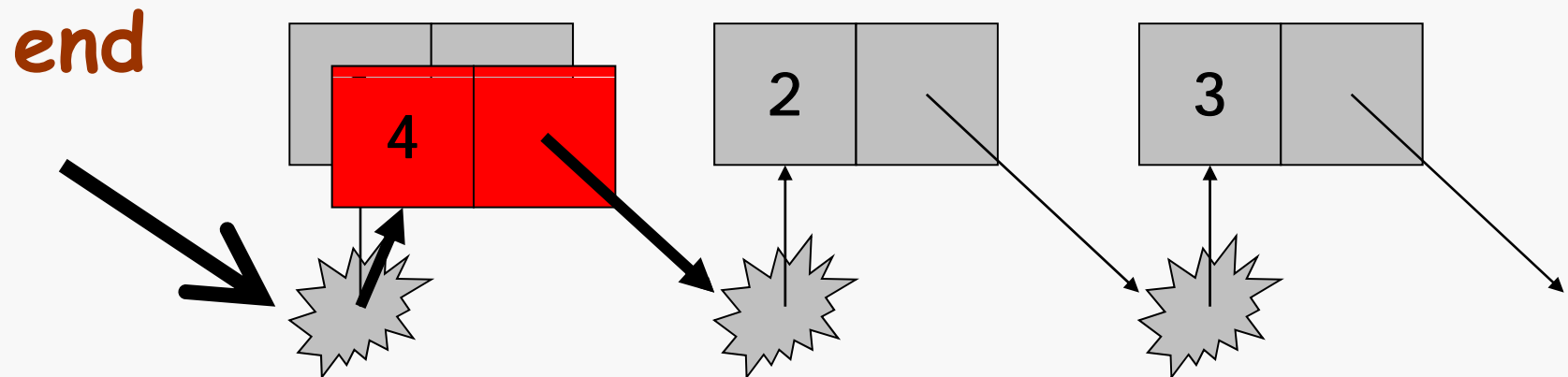
let setcar (Cons p) a = (fst(p) := a)

end



# Second Implementation

```
structure L'  
  type t = Nil | Cons of (int * t) ref  
  let nil = Nil  
  let cons a d = Cons(ref(a, d))  
  let setcar (Cons r) a =  
    r := (a, snd(!r))  
end
```



# The Problem

The implementations  $L$  and  $L'$  should be contextually equivalent under the interface  $S$

*How can we prove it?*

- Direct proof is infeasible because of the universal quantification: "for any context  $C$ "
- Little previous work deals with both abstract data types and references (cf. [Ahmed-Dreyer-Rossberg POPL'09])
  - None is complete (to my knowledge)

# Our Approach: Environmental Bisimulations

- Initially devised for  $\lambda$ -calculus with perfect encryption [Sumii-Pierce POPL'04]
- Successfully adapted for
  - Polymorphic  $\lambda$ -calculus [Sumii-Pierce POPL'05]
  - Untyped  $\lambda$ -calculus with references [Koutavas-Wand POPL'06] and deallocation [Sumii ESOP'09]
  - Higher-order  $\pi$ -calculus [Sangiorgi-Kobayashi-Sumii LICS'07]
  - Applied  $HO\pi$  [Sato-Sumii APLAS'09] etc.

# Our Target Language

Polymorphic  $\lambda$ -calculus with existential types and first-class references

$M ::= \dots$ standard  $\lambda$ -terms... |  
pack  $(\tau, M)$  as  $\exists\alpha.\sigma$  |  
open  $M$  as  $(\alpha, x)$  in  $N$  | locations  
ref  $M$  |  $!M$  |  $M := N$  |  $\ell$  |  $M == N$   
equality of locations

$\tau ::= \dots$ standard polymorphic types...  
|  $\exists\alpha.\tau$  |  $\tau$  ref

# Environmental Relations

An environmental relation  $X$  is a set of tuples of the form:

$$(\Delta, R, s \triangleright M, s' \triangleright M', \tau)$$

# Environmental Relations

An environmental relation  $X$  is a set of tuples of the form:

$$(\Delta, R, s \triangleright M, s' \triangleright M', \tau)$$

- Program  $M$  (resp.  $M'$ ) of type  $\tau$  is running under store  $s$  (resp.  $s'$ )
  - $M$  and  $M'$  (and  $\tau$ ) are omitted when terminated

# Environmental Relations

An environmental relation  $X$  is a set of tuples of the form:

$$(\Delta, \mathbf{R}, s \triangleright M, s' \triangleright M', \tau)$$

- Program  $M$  (resp.  $M'$ ) of type  $\tau$  is running under store  $s$  (resp.  $s'$ )
  - $M$  and  $M'$  (and  $\tau$ ) are omitted when terminated
- $\mathbf{R}$  is the environment: a (typed) relation between values known to the context

# Environmental Relations

An environmental relation  $X$  is a set of tuples of the form:

$$(\Delta, R, s \triangleright M, s' \triangleright M', \tau)$$

- Program  $M$  (resp.  $M'$ ) of type  $\tau$  is running under store  $s$  (resp.  $s'$ )
  - $M$  and  $M'$  (and  $\tau$ ) are omitted when terminated
- $R$  is the environment: a (typed) relation between values known to the context
- $\Delta$  maps an abstract type  $\alpha$  to (the pair of) their concrete types  $\sigma$  and  $\sigma'$

# Environmental Bisimulations for Our Calculus

An environmental relation  $X$  is an environmental bisimulation if it is preserved by

- execution of the programs and
- operations from the context

Formalized by the following conditions...

# Environmental Bisimulations: Condition for Reduction

- If  $(\Delta, R, s \triangleright M, s' \triangleright M', \tau) \in X$  and  $s \triangleright M$  converges to  $t \triangleright V$ , then  $s' \triangleright M'$  also converges to some  $t' \triangleright V'$  with  $(\Delta, R \cup \{(V, V', \tau)\}, t, t') \in X$

(Symmetric condition omitted)

Strictly speaking, this is a "big-step" version of environmental bisimulations

# Environmental Bisimulations: Condition for Opening

- If  $(\Delta, R, s, s') \in X$  and  
 $(\text{pack } (\tau, V) \text{ as } \exists\alpha.\sigma,$   
 $\text{pack } (\tau', V') \text{ as } \exists\alpha.\sigma, \exists\alpha.\sigma) \in R$ , then  
 $(\Delta \cup \{(\alpha, \tau, \tau')\}, R \cup \{(V, V', \sigma)\}, s, s') \in X$

# Environmental Bisimulations: Condition for Dereference

- If  $(\Delta, R, s, s') \in X$  and  $(l, l', \sigma \text{ ref}) \in R$ , then  $(\Delta, R \cup \{(s(l), s'(l'), \sigma)\}, s, s') \in X$

# Environmental Bisimulations: Condition for Update

- If  $(\Delta, R, s, s') \in X$  and  $(l, l', \sigma \text{ ref}) \in R$ , then  $(\Delta, R, s\{l \mapsto W\}, s'\{l' \mapsto W'\}) \in X$  for any  $W$  and  $W'$  synthesized from  $R$ 
  - Formally,

$$W = C[V_1, \dots, V_n]$$

$$W' = C[V'_1, \dots, V'_n]$$

for some  $(V_1, V'_1, \tau_1), \dots, (V_n, V'_n, \tau_n) \in R$   
and some well-typed  $C$

# Environmental Bisimulations: Condition for Application

- If  $(\Delta, R, s, s') \in X$  and  $(\lambda x.M, \lambda x.M', \sigma \rightarrow \tau) \in R$ , then  $(\Delta, R, s \triangleright [W/x]M, s' \triangleright [W'/x]M', \tau) \in X$  for any  $W$  and  $W'$  synthesized from  $R$

# Other Conditions

- Similar conditions for allocation, location equality, projection, etc.
- No condition for values of abstract types

If  $(\Delta, R, s, s') \in X$   
and  $(v, v', \alpha) \in R$ ,  
then ...?

- Context cannot operate on them

Abstract

# Mutable Integer Lists Interface (Reminder)

```
(* pseudo-code in  
  imaginary ML-like language *)  
signature S  
  type t (* abstract *)  
  val nil : t  
  val cons : int -> t -> t  
  val setcar : t -> int -> unit  
  (* setcdr, car, cdr, etc. *)  
end
```

# First Implementation (Reminder)

structure L

type t =

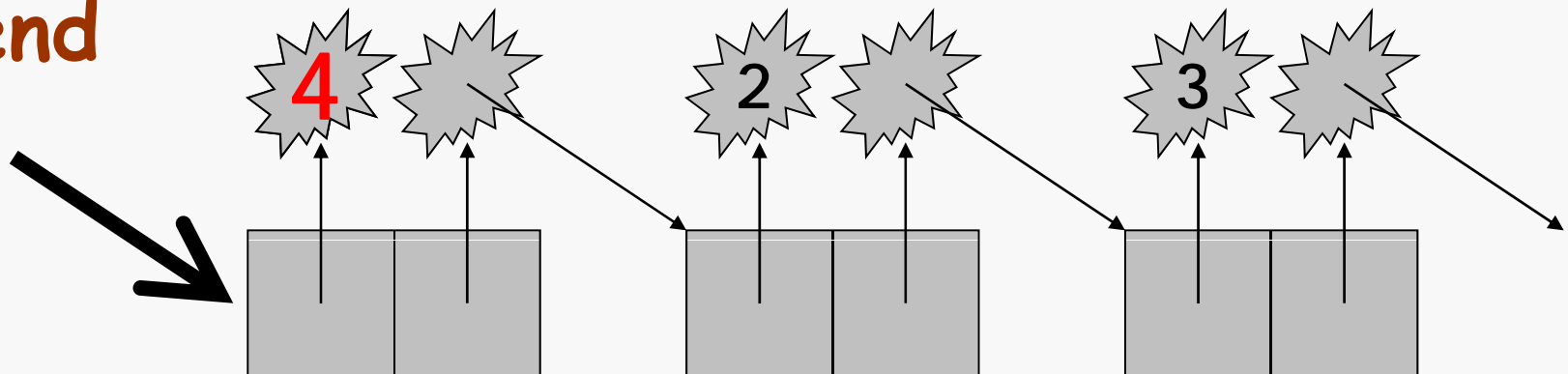
Nil | Cons of (int ref \* t ref)

let nil = Nil

let cons a d = Cons(ref a, ref d)

let setcar (Cons p) a = (fst(p) := a)

end



# Second Implementation (Reminder)

structure L'

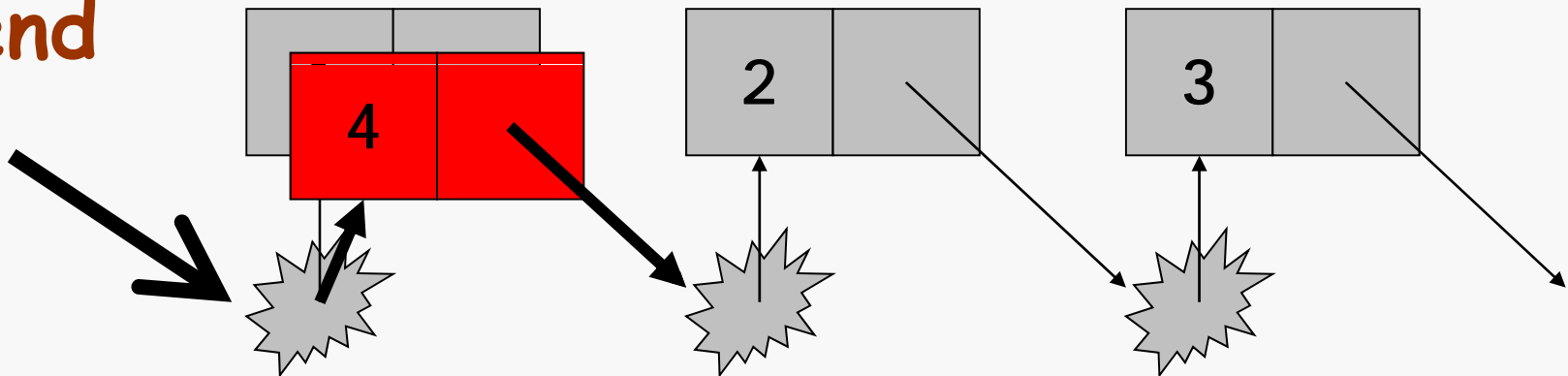
type t = Nil | Cons of (int \* t) ref

let nil = Nil

let cons a d = Cons(ref(a, d))

let setcar (Cons r) a =  
r := (a, snd(!r))

end



# Environmental Bisimulation for The Mutable Integer Lists

$$\begin{aligned} X &= \{ (\Delta, R, s, s') \mid \\ \Delta &= \{ (S.t, L.t, L'.t) \}, \\ R &= \{ (L, L', S), \\ &\quad (L.nil, L'.nil, S.t), \\ &\quad (L.cons, L'.cons, \text{int} \rightarrow S.t \rightarrow S.t), \\ &\quad (L.setcar, L'.setcar, S.t \rightarrow \text{int} \rightarrow \text{unit}), \\ &\quad (L.Cons(l_i, m_i), L'.Cons(l'_i), S.t) \\ &\quad (L.Nil, L'.Nil, S.t) \mid i = 1, 2, 3, \dots, n \}, \\ s(l_i) &= \text{fst}(s'(l'_i)) \text{ and} \\ (s(m_i), \text{snd}(s'(l'_i)), S.t) &\in R, \text{ for each } i \} \end{aligned}$$

# More complicated example (1/3)

(\* Adapted from [Ahmed-Dreyer-Rossberg  
POPL'09], credited to Thamsborg \*)

$\text{pack}(\text{int } \text{ref}, (\text{ref } 1, \lambda x.V_x))$  as  $\sigma$   
vs.  $\text{pack}(\text{int } \text{ref}, (\text{ref } 1, \lambda x.V'))$  as  $\sigma$

where

$V_x = \lambda f. (x := 0; f(); x := 1; f(); !x)$

$V' = \lambda f. (f(); f(); 1)$

$\sigma = \exists \alpha. \alpha \times (\alpha \rightarrow (1 \rightarrow 1) \rightarrow \text{int})$

- $f$  is supplied by the context
- What are the reducts of  $V f$  and  $V' f$ ?

# More complicated example (2/3)

$$X = X_0 \cup X_1$$

$X_0 = \{ (\Delta, R, \dagger\{l \mapsto 0\} \triangleright N, \dagger' \triangleright N', \text{int}) \mid$   
 $N$  and  $N'$  are made of contexts in  $T_0$ ,  
with holes filled with elements of  $R$  }

$X_1 = \{ (\Delta, R, \dagger\{l \mapsto 1\} \triangleright N, \dagger' \triangleright N', \text{int}) \mid$   
 $N$  and  $N'$  are made of contexts in  $T_1$ ,  
with holes filled with elements of  $R$  }

# More complicated example (3/3)

- $(C; l:=1; D; !l) T_0 (C; D; 1)$
- $(D; !l) T_1 (D; 1)$
- If  $E[zW] T_0 E'[zW]$ , then  
 $E[C; l:=1; D; !l] T_0 E'[C; D; 1]$   
(for any evaluation contexts  $E$  and  $E'$ )
- If  $E[zW] T_0 E'[zW]$ , then  $E[D; !l] T_1 E'[D; 1]$
- If  $E[zW] T_1 E'[zW]$ , then  
 $E[C; l:=1; D; !l] T_0 E'[C; D; 1]$
- If  $E[zW] T_1 E'[zW]$ , then  $E[D; !l] T_1 E'[D; 1]$

# Main Theorem: Soundness and Completeness

The largest environmental bisimulation  $\sim$   
coincides with (a generalized form of)  
contextual equivalence  $\equiv$

## Proof

- Soundness: Prove  $\sim$  is preserved under any context (by induction on the context)
- Completeness: Prove  $\equiv$  is an environmental bisimulation (by checking its conditions)

# The Caveat



Our "proof method" is not automatic

- Contextual equivalence in our language is undecidable
- Therefore, so is environmental bisimilarity

...but it proved all known examples!

# Up-To Techniques



Variants of environmental bisimulations  
with weaker (yet sound) conditions

- Up-to reduction (and renaming)
- Up-to context (and environment)
- Up-to allocation

Details in my CSL'09 paper

# Related Work

- Environmental bisimulations for other languages (already mentioned)
- Bisimulations for other languages
- Logical relations
- Game semantics

None has dealt with both abstract data types and references

- Except [Ahmed-Dreyer-Rossberg POPL'09]

# Conclusion of Part I

---

## Summary:

Sound and complete "proof method" for contextual equivalence in polymorphic  $\lambda$ -calculus with existential types and references

## Current and future work:

- Parametricity properties ("free theorems")
- Semantic model

**Part II: Enviromental bisimulations  
for higher-order  $\pi$ -calculus  
with encryption**

**Nobuyuki Sato**

**Eijiro Sumii**

**(Tohoku University)**



# Agenda of Part II



- **Informal overview of the work**
- **A little technical details**
- **A little more technical details**
- **Conclusion**

# Main Result



A bisimulation proof technique  
for higher-order process calculus  
with cryptographic primitives

- Can be used for proving security properties of concurrent systems that send/receive programs using encryption/decryption

# Motivation

Higher-order cryptographic systems  
are now ubiquitous

- Web-based e-mail clients (e.g. Gmail)
- Software update systems (e.g. Windows Update)

Higher-order: transmitting programs themselves

⇒ Security is even more important  
than in first-order systems

- Cryptography is essential

# Problem



The theory of  
higher-order cryptographic  
computation is underdeveloped

- Little work for the combination of higher-order processes and cryptographic primitives  
Cf. Higher-order pi-calculus (no cryptography), spi-calculus (first-order), ...

# A Challenge of Higher-Order Cryptographic Processes

- Consider the process  $P = \bar{c}\langle Q \rangle$   
where  $Q = \bar{c}\langle \text{encrypt}(m, k) \rangle$ 
  - $\bar{c}\langle \rangle$  denotes output to the network  $c$
  - Assume  $c$  is public and  $k$  is secret
- Does  $P$  leak  $m$ ?
  1. Yes, because the attacker can receive  $Q$  from  $c$  and extract  $m$
  2. No, if  $m$  is encrypted before  $Q$  is sent to  $c$

# Observations

- Computation (e.g. encryption) and computed values (e.g. ciphertext) must be distinguished
- The attacker should be able to decompose transmitted processes (but not computed values)

(Recall the previous example  $P = \bar{c}\langle Q \rangle$   
where  $Q = \bar{c}\langle \text{encrypt}(m, k) \rangle$ )

# Solution

- Syntactically distinguish computation (e.g. `encrypt(m,k)`) and computed values (e.g. `^encrypt(m,k)`)
- Extend the calculus with a primitive to decompose transmitted processes:  
`match P as x in Q`  
(bind `x` to the decomposed abstract syntax tree of `P` and execute `Q`)
  - Computed values can not be decomposed

# Examples

$\bar{c} \langle \bar{c} \langle \text{encrypt}(m, k) \rangle \rangle \mid$

$c(X). \text{match } X \text{ as } y \text{ in } R$

→  $\text{match } \bar{c} \langle \text{encrypt}(m, k) \rangle \text{ as } y \text{ in } R$

→  $[\text{Out}(\text{Nam } c, \text{Enc}(\text{Nam } m, \text{Nam } k)) / y] R$

$\bar{c} \langle \bar{c} \langle \hat{\text{encrypt}}(m, k) \rangle \rangle \mid$

$c(X). \text{match } X \text{ as } y \text{ in } R$

→  $\text{match } \bar{c} \langle \hat{\text{encrypt}}(m, k) \rangle \text{ as } y \text{ in } R$

→  $[\text{Out}(\text{Nam } c, \text{Val } \hat{\text{encrypt}}(m, k)) / y] R$

# Next Challenge

How do we reason about higher-order cryptographic processes?

- Traditional techniques (bisimulations, in particular) do not apply
  - Most of them are first-order
  - Normal bisimulations [Sangiorgi 92] are unsound for process decomposition
    - Because they only transmit "triggers" (i.e. pointers to processes)

# Solution



## Adopt environmental bisimulations

- Devised for  $\lambda$ -calculus with encryption [Sumii-Pierce 04]
- Adapted for various languages [Sumii-Pierce, Koutavas-Wand, ...]
  - Including higher-order pi-calculus [Sangiorgi-Kobayashi-Sumii 07]

# Idea of Environmental Bisimulations

- Traditional (i.e. non-environmental) bisimulation  $P \sim P'$  means:
  - $P$  and  $P'$  behave the same under any observer process
- Environmental bisimulation  $P \sim_E P'$  means:
  - $P$  and  $P'$  behave the same under any observer process that uses any elements  $(V, V')$  of  $E$
  - $E$  is a binary relation on values that represents the observer's knowledge (called an environment)

# Agenda of Part II



- Informal overview of the work
- **A little technical details**
- **A little more technical details**
- **Conclusion**

# Our Environmental Bisimulations (1/3)

Binary relation  $X$  on processes,  
indexed by environments  $E$ ,  
is an environmental simulation  
if  $P X_E P'$  implies:

1. If  $P$  reduces to  $Q$ , then  
 $P'$  reduces to some  $Q'$   
such that  $Q X_E Q'$
2. If  $P$  outputs  $V$  and becomes  $Q$ , then  
 $P'$  outputs some  $V'$  and becomes some  $Q'$   
such that  $Q X_{E \cup \{(V, V')\}} Q'$

(cont.)

# Our Environmental Bisimulations (2/3)

$X$  is an environmental simulation  
if  $P X_E P'$  implies:

3. For any  $V$  and  $V'$  composed from  $E$ ,  
if  $P$  inputs  $V$  and becomes  $Q$ , then  
 $P'$  inputs  $V'$  and becomes some  $Q'$   
such that  $Q X_E Q'$ 
  - "Composed from" means  
for some context  $C$  and  $(V_1, V_1'), \dots, (V_n, V_n') \in E$ ,  
 $V = C[V_1, \dots, V_n]$  and  $V' = C[V_1', \dots, V_n']$
4.  $P|Q X_E P'|Q'$  for any  $(Q, Q') \in E$

(cont.)

# Our Environmental Bisimulations (3/3)

$X$  is an environmental simulation  
if  $P X_E P'$  implies:

5.  $P X_{E \cup \{(V, V')\}} P'$  if  $V$  and  $V'$  can be computed from  $E$  (by decomposition or first-order computation)

E.g. suppose:

$$E = \{(k, k'), (\hat{\text{encrypt}}(V, k), \hat{\text{encrypt}}(V', k'))\}$$

Then  $(V, V')$  can be computed from  $E$   
by the first-order context:

$$C = \text{decrypt}([\ ]_2, [\ ]_1)$$

6.  $E$  preserves equality

# Main Theorem

The largest environmental bisimulation (with appropriate  $E$ ) coincides with reduction-closed barbed equivalence

- It exists because the generating function is monotone [Tarski 55]
- The  $\subseteq$  direction is proved via a context closure property of environmental bisimulations
- The  $\supseteq$  direction is proved by coinduction

# Agenda of Part II



- Informal overview of the work
- A little technical details
- **A little more technical details**
- **Conclusion**

# Our Calculus: Syntax of Terms

$M ::=$	terms
$V$	values
$x$	variables
$M(M_1, \dots, M_n)$	computations
$V ::=$	values
$a$	names
$f$	function symbols
$\hat{f}(V_1, \dots, V_n)$	computed values
$`P$	transmitted processes
$`M$	transmitted terms

# Syntax of Processes

$P ::=$	processes
$0$	inaction
$M(x).P$	input
$\bar{M}\langle N \rangle.P$	output
$P Q$	parallel composition
$!P$	replication
$\nu x.P$	restriction
$\text{run}(M)$	execution
$\text{if } M=N \text{ then } P \text{ else } Q$	conditional
$\text{match } M \text{ as } x \text{ in } P$	decomposition

# Labeled Transition Semantics

- Parameterized by semantics of terms
  - Defined by (strongly normalizing and confluent) term rewriting system

- Key rules:

$$\overline{c\langle M \rangle}.P \xrightarrow{\overline{c\langle V \rangle}} P$$

if  $M$  rewrites to  $V$  ("call-by-value")

$$\text{run}(P) \xrightarrow{\tau} P \quad (\text{important!})$$

$$\text{match } P \text{ as } x \text{ in } Q \xrightarrow{\tau} [M/x]Q$$

where  $M$  is decomposed AST of  $P$

# Examples (Revisited)

$\bar{c} \langle \text{ ` } \bar{c} \langle \text{encrypt}(m, k) \rangle \text{ `} \rangle \mid$

$c(X).match\ X\ \text{as}\ y\ \text{in}\ R$

→  $match\ \text{ ` } \bar{c} \langle \text{encrypt}(m, k) \rangle \text{ `} \text{ as}\ y\ \text{in}\ R$

→  $[Out(Nam\ c, Enc(Nam\ m, Nam\ k))/y]R$

$\bar{c} \langle \text{ ` } \bar{c} \langle \hat{\text{encrypt}}(m, k) \rangle \text{ `} \rangle \mid$

$c(X).match\ X\ \text{as}\ y\ \text{in}\ R$

→  $match\ \text{ ` } \bar{c} \langle \hat{\text{encrypt}}(m, k) \rangle \text{ `} \text{ as}\ y\ \text{in}\ R$

→  $[Out(Nam\ c, Val\ \hat{\text{encrypt}}(m, k))/y]R$

# Bisimulation Example

$P = \bar{c} \langle \text{c} \langle \hat{\text{encrypt}}(3, k) \rangle \rangle$  and  
 $P' = \bar{c} \langle \text{c} \langle \hat{\text{encrypt}}(7, k) \rangle \rangle$   
are bisimilar

Proof outline: Take  $X$  as follows (so  $P X_E P'$ )

$X = \{ (E, C[\hat{\text{encrypt}}(3, k)], C[\hat{\text{encrypt}}(7, k)]) \mid$   
 $k \text{ not free in } C \}$

$E = \{ (D[\hat{\text{encrypt}}(3, k)], D[\hat{\text{encrypt}}(7, k)]) \mid$   
 $k \text{ not free in } D \}$

and prove it to be an env. bisim.  
(by case analysis on  $C$  and  $D$ )

# Non-Bisimulation Example

$P = \bar{c} \langle \bar{c} \langle \text{encrypt}(3, k) \rangle \rangle$  and  
 $P' = \bar{c} \langle \bar{c} \langle \text{encrypt}(7, k) \rangle \rangle$  are  
not bisimilar

Proof outline:

If  $P \sim_E P'$  for some env. bisim.  $X$  and  $E$ ,  
then by output we get  $0 \sim_{E'} 0$  with  
 $(\bar{c} \langle \text{encrypt}(3, k) \rangle, \bar{c} \langle \text{encrypt}(7, k) \rangle) \in E'$ .

Since  $(3, 7)$  can be computed from  $E'$  by  
decomposition, we get  $0 \sim_{E''} 0$  with  
 $(3, 7) \in E''$ , which violates integer equality.

# Simplification by Up-To Context Technique

Problem:

Many environmental bisimulations include all processes/values of the forms

$C[V_1, \dots, V_n]$  and  $C[V_1', \dots, V_n']$   
for some  $(V_1, V_1'), \dots, (V_n, V_n')$

Solution:

A "smaller" version of environmental bisimulations, where processes/values of the forms  $C[V_1, \dots, V_n]$  and  $C[V_1', \dots, V_n']$  can be omitted if  $(V_1, V_1'), \dots, (V_n, V_n')$  are included

# Example of Environmental Bisimulation Up-To Context

Consider again:

$$P = \bar{c} \langle \cdot, \bar{c} \langle \hat{\text{encrypt}}(3, k) \rangle \rangle$$

$$P' = \bar{c} \langle \cdot, \bar{c} \langle \hat{\text{encrypt}}(7, k) \rangle \rangle$$

Then

$$Y = \{ (E, P, P') \}$$

is an environmental bisimulation up-to context, where:

$$E = \{(c, c), (\hat{\text{encrypt}}(3, k), \hat{\text{encrypt}}(7, k))\}$$

# In our APLAS'09 paper

- Formal definitions of the calculus and our environmental bisimulations (and the up-to context technique)
- Soundness and completeness proofs (i.e. proof of coincidence with reduction-closed barbed equivalence)
- More sophisticated examples
  - Software distribution system
  - Online e-mail client

# Agenda of Part II



- Informal overview of the work
- A little technical details
- A little more technical details
- **Conclusion**

# Conclusions of Part II

- Higher-order cryptographic processes are non-trivial
  - Previous theories do not apply (higher-order pi-calculus, spi-calculus, ...)
- Environmental bisimulations "scale" well to such sophisticated calculi
  - Including the present one
- Future work:  
automation, extension, simplification, ...

# Sources of Discussion/Controversy

- Relationship to denotational semantics (especially, games)
  - Denotational semantics is "syntax-free"
  - Env. bisim. is "semantics-free"
    - Very robust, but (arguably) ugly, lacking good mathematical structure
    - ⇒ More structured and robust framework?
- (Semi-)automation/mechanization
  - What does "completeness" mean when the problem is undecidable?