Theories of Information Hiding in Lambda-Calculus

Logical Relations and Bisimulations for Encryption and Type Abstraction

Eijiro Sumii

University of Pennsylvania

Penn University of Pennsylvania

東京大学 THE UNIVERSITY OF TOKYO

Main Results

- Proof methods for two forms of information hiding in computer programs
  - Logical relations for perfect encryption
  - Bisimulations for perfect encryption
  - Bisimulations for type abstraction
    - First solution to a problem of 20 years

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Background

- Information hiding (or abstraction) is crucial for building large systems
  - ...including computer software!
- Type abstraction is the primary method of information hiding in programming languages
  - The basis of objects, modules, components, etc.

A Classical Example

```
interface Complex =
  type t
  fun make : real ´ real ® t
  fun mul : t ´ t ® t
  fun re : t ® real
```

Penn University of Pennsylvania

東京大学 THE UNIVERSITY OF TOKYO

Cartesian Implementation

```
module Cartesian : Complex =
  type t = real ´ real
  fun make(x,y) = (x,y)
  fun mul((x1,y1),(x2,y2)) =
    (x1*x2-y1*y2 , x1*y2+x2*y1)
  fun re(x,y) = x
```

Penn University of Pennsylvania
東京大学 THE UNIVERSITY OF TOKYO

Polar Implementation

```
module Polar : Complex =
  type t = real ´ real
  fun make(x,y) =
    (sqrt(x*x+y*y),atan2(y,x))
  fun mul((r1,q1),(r2,q2)) =
    (r1*r2,q1+q2)
  fun re(r,q) = r*cos(q)
```

Penn
University of Pennsylvania
東京大学
THE UNIVERSITY OF TOKYO

The Abstraction Property

Contextual equivalence:

$\vdash$ `Cartesian` ° `Polar` : `Complex`

I.e., the different implementations give the same result under any well-typed context in the language

- **In this study, "result" means only the final value (or possible divergence)**
  - **Time, energy, rounding errors, etc. are out of scope**

Penn
UNIVERSITY of PENNSYLVANIA

東京大学
THE UNIVERSITY OF TOKYO

How to Prove it?

<u>Logical relations</u> [Reynolds 83]:

Induction on the type of the interface (`Complex` in previous example)

- So far, so good.

The Problems

- Logical relations become complex for expressive languages (e.g. with recursion or concurrency)
  - ...yet <u>attackers must be expressed in the language</u> (as well as users)
- Type abstraction doesn't work against untyped users/attackers
  - ...but we cannot "type-check the Internet"

Penn
UNIVERSITY of PENNSYLVANIA

東京大学
THE UNIVERSITY OF TOKYO

Our Solutions

- Use <u>bisimulations</u> in place of logical relations
- Use <u>encryption</u> instead of type abstraction

Outline

1. **Logical relations for type abstraction**
   [Reynolds 83] [Mitchell 91]
2. **Bisimulations for type abstraction**
   [Sumii-Pierce, POPL'05]
3. **Bisimulations for encryption**
   [Sumii-Pierce, POPL'04 & TCS]
   - **Cf. Logical relations for encryption**
     [Sumii-Pierce, CSFW'01 & JCS]

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Logical Relations (without Type Abstraction)

Relations between programs, defined by induction on their types

- Constants are related iff they are equal
- Tuples are related iff their elements are related
- Functions are related iff they map related arguments to related results

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Examples

- **123** and **123** are related at type **Int**
- **lx.x+1+2** and **lx.x+3** are related at type **Int®Int**
- **lx.(123,x+1+2)** and **lx.(123,x+3)** are related at type **Int®(Int´Int)**

Penn
UNIVERSITY of PENNSYLVANIA

東京大学
THE UNIVERSITY OF TOKYO

Logical Relations for Type Abstraction

In addition to the previous cases:

- **Abstract data of type a are related iff they are related by $j(a)$**
  - **where j is a <u>relation environment</u> mapping abstract types to the relation between their implementations**

Example

Take

$j($**`Complex.t`**$) =$

$\{((\texttt{x},\texttt{y}),(\texttt{r},q)) \mid x = r \cos q,\ y = r \sin q\}$

Then

$j \vdash$ **`Cartesian`** $\sim$ **`Polar`** : **`Complex`**

That is, `Cartesian` and `Polar` are logically related at type `Complex` under j

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Soundness of Logical Relations

Logical relations imply contextual equivalence

- Corollary of the "fundamental property" (a.k.a. the "basic lemma")
  - Proved by induction on the typing of terms, with assumptions on their free variables

Shortcoming of Logical Relations

Don't "scale" to more expressive languages

- **Recursive functions (or while-loops) complicate the fundamental property**
- **Recursive <u>types</u> complicate the <u>definition</u> of logical relations**

Extra work required in <u>every use</u> of logical relations (cannot be done once and for all in meta theory)

Outline

1. Logical relations for type abstraction
   [Reynolds 83] [Mitchell 91]
2. Bisimulations for type abstraction
   [Sumii-Pierce, POPL'05]
3. Bisimulations for encryption
   [Sumii-Pierce, POPL'04 & TCS]

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Bisimulations (without Type Abstraction)

Applicative bisimulations [Abramsky 90]:

Relations between values that satisfy some conditions to exclude inequivalent values

- Bisimilar constants are equal
- Bisimilar tuples have bisimilar elements
- Bisimilar functions return bisimilar results when applied to the same argument

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Examples

- {(123, 123)} is a bisimulation
- {(4, 4), (5, 5)} is a bisimulation
- {((4,5),(4,5)), (4, 4), (5, 5)} is a bisimulation
- {(lx.x+1+2, lx.x+3), (i, j) | i = k+1+2, j = k+3, k : int} is a bisimulation
- Union of bisimulations is a bisimulation

Shortcoming of Applicative Bisimulations

Don't extend to type abstraction

- `Cartesian.re` and `Polar.re` do not return bisimilar results (i.e. the same real number) when applied to the same argument
  - They expect different representations

Penn University of Pennsylvania

東京大学 THE UNIVERSITY OF TOKYO

Bisimulations for Type Abstraction: First Try

Bisimilar functions return bisimilar results <u>when applied to bisimilar arguments</u>

- **No condition for abstract data themselves, as long as the other conditions are satisfied**

<u>THIS IS UNSOUND!</u>

- **Because contexts (users or attackers) can combine bisimilar values to make more complex arguments**

Penn UNIVERSITY of PENNSYLVANIA

東京大学 THE UNIVERSITY OF TOKYO

Counter-Example

- `(1,fst)` and `(2,fst)` are <u>not</u> contextually equivalent at type a ´ (a ´ a ® `int`)

But

{((`1`,`fst`),(`2`,`fst`), a ´(a ´ a ® `int`)),
(`1`, `2`, a),
(`fst`, `fst`, a ´ a ® `int`)}

satisfies all the bisimulation conditions so far!

Penn
UNIVERSITY of PENNSYLVANIA

東京大学
THE UNIVERSITY OF TOKYO

Bisimulations for Type Abstraction: Second Try

Bisimilar functions return bisimilar results when applied to $C[v_1,\ldots,v_n]$ and $C[v_1',\ldots,v_n']$

- for any bisimilar values $v_1,\ldots,v_n$ and $v_1',\ldots,v_n'$, and
- for any value context C.

Penn UNIVERSITY of PENNSYLVANIA

東京大学 THE UNIVERSITY OF TOKYO

Example

```
R = { (Cartesian, Polar, Complex),
      (Cartesian.make, Polar.make,
       real ´ real ® Complex.t),
      (Cartesian.mul, Polar.mul,
       Complex.t ´ Complex.t ® Complex.t),
      (Cartesian.re, Polar.re,
       Complex.t ® real),
      ((x,y),(r,q), Complex.t),
      (z, z, real) | x = r cos q, y = r sin q }
```

Penn University of Pennsylvania

東京大学 THE UNIVERSITY OF TOKYO

The Last Problem

The previous definition is sound, but completeness is unclear

- **Because the union of two bisimulations wouldn't always be a bisimulation**
  - **Counter-example: the union RÈR^{-1} of the previous R and its inverse R^{-1}**

Þ **Standard co-inductive method wouldn't apply**

Penn University *of* Pennsylvania

東京大学 The University of Tokyo

Our Solution

Consider sets of relations as bisimulations

- Intuition: Each relation represents a "world"
- Another intuition: Each relation represents the knowledge of an attacker, which increases by time (but nevertheless stays in the bisimulation)
- Also gives a natural account for the generativity of existential types

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Examples

For the previous R,

- **{ R } is a bisimulation**
- **{ R^{-1} } is another bisimulation**
- **{ R, R^{-1} } is also a bisimulation**
- **{ RÈR^{-1} } is <u>not</u> a bisimulation**

Soundness and Completeness

- Contextual equivalence is also generalized as a set of relations
- Then, it coincides with <u>bisimilarity</u> (the largest bisimulation)
  - Everything is formalized and proved in l-calculus with full universal, existential, and recursive types (first result in 20 years!)

Penn
University of Pennsylvania
東京大学
The University of Tokyo

Other Examples

- Object encodings
- ML-like functors
- Higher-order polymorphic functions (the "dual" of abstract types)

Outline

1. Logical relations for type abstraction
   [Reynolds 83] [Mitchell 91]
2. Bisimulations for type abstraction
   [Sumii-Pierce, POPL'05]
3. **Bisimulations for encryption**
   [Sumii-Pierce, POPL'04 & TCS]

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Idea: Abstraction by Encryption

Reinvention of dynamic sealing [Morris 73]

- Secret key is generated for each abstract type
- Abstract data are encrypted when exported out of a module
- ...and decrypted when imported back

Cartesian Implementation of `Complex` with Encryption

```
module Cartesian =
  fun make(x,y) = encrypt_k(x,y)
  fun mul(c_1,c_2) =
   let (x_1,y_1) = decrypt_k(c_1) in
   let (x_2,y_2) = decrypt_k(c_2) in
   encrypt_k(x_1*x_2-y_1*y_2 , x_1*y_2+x_2*y_1)
  fun re(c) =
   let (x,y) = decrypt_k(c) in x
```

Penn University of Pennsylvania
東京大学 The University of Tokyo

Polar Implementation of Complex with Encryption

```
module Polar =
  fun make(x,y) =
   encrypt_k'(sqrt(x*x+y*y),atan2(y,x))
  fun mul(c_1,c_2) =
   let (r_1,q_1) = decrypt_k'(c_1) in
   let (r_2,q_2) = decrypt_k'(c_2) in
   encrypt_k'(r_1*r_2,q_1+q_2)
  fun re(c) =
   let (r,q) = decrypt_k'(c) in r*cos(q)
```

Penn University of Pennsylvania
東京大学 THE UNIVERSITY OF TOKYO

The Abstraction Property

Untyped contextual equivalence:

`Cartesian` o `Polar`

- **Abstraction holds against any context in the language, <u>even if untyped</u>**

How to Prove it?

Bisimulations!

- **Conditions for constants, tuples, functions are the same as before**

Bisimulations for Encryption

- Bisimulation respects equality of keys
  - I.e., if k_1 and k_1' are bisimilar, and if k_2 and k_2' are bisimilar, then $k_1 = k_2$ Û $k_1' = k_2'$
- For any bisimilar ciphertexts $\mathtt{encrypt}_k(\mathtt{v})$ and $\mathtt{encrypt}_{k'}(\mathtt{v'})$,
  - Neither k nor k' is in the relation, or
  - v and v' are bisimilar

Example

R = { (Cartesian, Polar),

(Cartesian.make, Polar.make),

(Cartesian.mul, Polar.mul),

(Cartesian.re, Polar.re),

($\mathrm{encrypt}_k$(x,y), $\mathrm{encrypt}_{k'}$(r,q)),

(z, z) |

x = r cos q, y = r sin q, z : real }

Penn University of Pennsylvania
東京大学 THE UNIVERSITY OF TOKYO

Formalization

- Defined untyped λ-calculus extended with:
  - Keys **k** and fresh key generation **new x in e**
  - Encryption $\{e_1\}_{e_2}$ and decryption $\text{let } \{x\}_{e_1} = e_2 \text{ in } e_3 \text{ else } e_4$
    - Assumes perfect encryption
- Proved soundness and completeness of our bisimulations in this language

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Operational Semantics of the Language (1/2)

- Big-step evaluation (s)e ß (t)v from terms e to values v
  - Annotated with the set of keys s and t before and after the evaluation

$$\frac{\text{k Ï s} \qquad \text{(sÈ\{k\}) [k/x]e ß (t)v}}{\text{(s) new x in e ß (t)v}}$$

Penn University of Pennsylvania
東京大学 THE UNIVERSITY OF TOKYO

Operational Semantics of the Language (2/2)

- **Success of decryption:**

$$\frac{(s)e_1 \text{ ß } (s_1)k \qquad (s_1)e_2 \text{ ß } (s_2)\{v\}_k \qquad (s_2)\ [v/x]e_3 \text{ ß } (t)w}{(s)\ \mathrm{let}\ \{x\}_{e_1} = e_2\ \mathrm{in}\ e_3\ \mathrm{else}\ e_4 \text{ ß } (t)w}$$

- **Failure of decryption:**

$$\frac{(s)e_1 \text{ ß } (s_1)k \qquad (s_1)e_2 \text{ ß } (s_2)\{v\}_{k'} \qquad k \text{ ¹ } k' \qquad (s_2)e_4 \text{ ß } (t)w}{(s)\ \mathrm{let}\ \{x\}_{e_1} = e_2\ \mathrm{in}\ e_3\ \mathrm{else}\ e_4 \text{ ß } (t)w}$$

Penn University of Pennsylvania
東京大学 THE UNIVERSITY OF TOKYO

Other Examples

- Generative functors
- Non-generative functors
- Encodings of security protocols

Our Protocol Encoding

- A protocol is encoded as a tuple of participants (and their public keys)
  - Senders are encoded as the values being sent
  - Receivers are encoded as functions from received values to returned values
- Then, <u>contexts play the role of the network, scheduler, and attackers</u> by applying the receivers to the senders

Penn University of Pennsylvania

東京大学 THE UNIVERSITY OF TOKYO

Example

1. A ® B : $\{N\}_K$
2. B ® A : N mod 2

Sys_N = **new x in**

(encrypt$_x$(N),

l y.(decrypt$_x$(y) mod 2))

THE UNIVERSITY OF TOKYO 東京大学

Penn UNIVERSITY of PENNSYLVANIA

Non-Interference: Secrecy as Equivalence

With our bisimulations, it is easy to prove

$$\mathbf{Sys_M} \circ \mathbf{Sys_N}$$

for any M and N with

$$\mathbf{M \bmod 2 = N \bmod 2}$$

Which means:

Sys_N keeps N secret (except for its least significant bit) under any context

Example

$$
\begin{aligned}
R = \{\, & (\mathtt{Sys}_M,\ \mathtt{Sys}_N),\\
& (\mathtt{encrypt}_k(M),\ \mathtt{encrypt}_{k'}(N)),\\
& (\lambda y.(\mathtt{decrypt}_k(y)\ \mathtt{mod}\ 2),\\
& \ \lambda y.(\mathtt{decrypt}_{k'}(y)\ \mathtt{mod}\ 2)),\\
& (M \bmod 2,\ N \bmod 2)\,\}
\end{aligned}
$$

Penn
University *of* Pennsylvania

東京大学
The University of Tokyo

Protocols Encoded and Proved (or Disproved)

- Needham-Schroeder (insecure)
- Needham-Schroeder-Lowe (secure)
- "ffgg" protocol [Millen 99] (insecure)
  - Attack is "necessarily parallel" but can be simulated in l-calculus via interleaving

Various properties (such as integrity) can be checked as long as expressed as equivalence

Needham-Schroeder-Lowe Protocol

$$
\begin{array}{llll}
1. & \mathsf{B} \to \mathsf{A} & : & B \\
2. & \mathsf{A} \to \mathsf{B} & : & \{N_A, A\}_{k_B} \\
3. & \mathsf{B} \to \mathsf{A} & : & \{N_A, N_B, B\}_{k_A} \\
4. & \mathsf{A} \to \mathsf{B} & : & \{N_B\}_{k_B} \\
5. & \mathsf{B} \to \mathsf{A} & : & \{i\}_{N_B}
\end{array}
$$

Penn
University *of* Pennsylvania

東京大学
The University of Tokyo

Encoding of Needham-Schroeder-Lowe Protocol

$$
\begin{aligned}
W &= \langle \lambda x.\{x\}_{k_A}, \lambda x.\{x\}_{k_B}, k_E, U, V \rangle \\[1em]
U &= \langle B, \lambda\{\langle x, y\rangle\}_{k_B}.\, \mathtt{assert}(y = A); \\
&\qquad\qquad\qquad \nu z.\, \langle \{\langle x, z, B\rangle\}_{k_A}, \\
&\qquad\qquad\qquad\qquad \lambda\{z_0\}_{k_B}.\, \mathtt{assert}(z_0 = z); \\
&\qquad\qquad\qquad\qquad\qquad \{i\}_z \rangle\rangle \\
V &= \lambda x.\, \mathtt{let}\ k_x = (\mathtt{if}\ x = B\ \mathtt{then}\ k_B\ \mathtt{else} \\
&\qquad\qquad\qquad \mathtt{if}\ x = E\ \mathtt{then}\ k_E\ \mathtt{else}\ \bot)\ \mathtt{in} \\
&\qquad \nu y.\, \langle \{(y, A)\}_{k_x}, \\
&\qquad\qquad \lambda\{\langle y_0, z, x_0\rangle\}_{k_A}.\, \mathtt{assert}(y_0 = y); \\
&\qquad\qquad\qquad \mathtt{assert}(x_0 = x); \\
&\qquad\qquad\qquad \{z\}_{k_x} \rangle
\end{aligned}
$$

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Bisimulation for Needham-Schroeder-Lowe Protocol

$$
\begin{array}{l}
\{(\mathsf{U},\mathsf{U}'),(\mathsf{V},\mathsf{V}'),(\mathsf{W},\mathsf{W}'),\\
\ (\overline{k},\overline{k}'),(A,A),(B,B),(E,E),(\lambda x.\{x\}_{k_A},\lambda x.\{x\}_{k'_A}),(\lambda x.\{x\}_{k_B},\lambda x.\{x\}_{k'_B}),(k_E,k'_E),(\overline{w},\overline{w}'),(\{\overline{w}\}_{k_A},\{\overline{w}'\}_{k'_A}),(\{\overline{w}\}_{k_B},\{\overline{w}'\}_{k'_B}),\\
\ (\lambda\{\langle x,y\rangle\}_{k_B}.\mathtt{assert}(y=A);\nu z.(\{\langle x,z,B\rangle\}_{k_A},\lambda\{z_0\}_{k_B}.\mathtt{assert}(z_0=z);\{i\}_z),\ \lambda\{\langle x,y\rangle\}_{k'_B}.\mathtt{assert}(y=A);\nu z.(\{\langle x,z,B\rangle\}_{k'_A},\lambda\{z_0\}_{k'_B}.\mathtt{assert}(z_0=z);\{j\}_z)),\\
\ (\langle\{\langle\overline{k}_{AB},A\rangle\}_{k_B},\lambda\{\langle y_0,z,x_0\rangle\}_{k_A}.\mathtt{assert}(y_0=\overline{k}_{AB});\mathtt{assert}(x_0=B);\{z\}_{k_B}\rangle,\ \langle\{\langle\overline{k}'_{AB},A\rangle\}_{k'_B},\lambda\{\langle y_0,z,x_0\rangle\}_{k'_A}.\mathtt{assert}(y_0=\overline{k}'_{AB});\mathtt{assert}(x_0=B);\{z\}_{k'_B}\rangle),\\
\ (\{\langle\overline{k}_{AB},A\rangle\}_{k_B},\{\langle\overline{k}'_{AB},A\rangle\}_{k'_B}),\\
\ (\lambda\{\langle y_0,z,x_0\rangle\}_{k_A}.\mathtt{assert}(y_0=\overline{k}_{AB});\mathtt{assert}(x_0=B);\{z\}_{k_B},\ \lambda\{\langle y_0,z,x_0\rangle\}_{k'_A}.\mathtt{assert}(y_0=\overline{k}'_{AB});\mathtt{assert}(x_0=B);\{z\}_{k'_B}),\\
\ (\langle\{\langle\overline{k}_{AB},\overline{k}_B,B\rangle\}_{k_A},\lambda\{z_0\}_{k_B}.\mathtt{assert}(z_0=\overline{k}_{AB});\{i\}_{\overline{k}_B}\rangle,\ \langle\{\langle\overline{k}'_{AB},\overline{k}'_B,B\rangle\}_{k'_A},\lambda\{z_0\}_{k'_B}.\mathtt{assert}(z_0=\overline{k}'_{AB});\{j\}_{\overline{k}'_B}\rangle),\\
\ (\{\langle\overline{k}_{AB},\overline{k}_B,B\rangle\}_{k_A},\{\langle\overline{k}'_{AB},\overline{k}'_B,B\rangle\}_{k'_A}),\\
\ (\lambda\{z_0\}_{k_B}.\mathtt{assert}(z_0=\overline{k}_{AB});\{i\}_{\overline{k}_B},\ \lambda\{z_0\}_{k'_B}.\mathtt{assert}(z_0=\overline{k}'_{AB});\{j\}_{\overline{k}'_B}),\\
\ (\{\overline{k}_B\}_{k_B},\{\overline{k}'_B\}_{k'_B}),\\
\ (\{i\}_{\overline{k}_B},\{j\}_{\overline{k}'_B}),\\
\ (\langle\{\langle\overline{k}_{AE},A\rangle\}_{k_E},\lambda\{\langle y_0,z,x_0\rangle\}_{k_A}.\mathtt{assert}(y_0=\overline{k}_{AE});\mathtt{assert}(x_0=E);\{z\}_{k_E}\rangle,\ \langle\{\langle\overline{k}'_{AE},A\rangle\}_{k'_E},\lambda\{\langle y_0,z,x_0\rangle\}_{k'_A}.\mathtt{assert}(y_0=\overline{k}'_{AE});\mathtt{assert}(x_0=E);\{z\}_{k'_E}\rangle),\\
\ (\{\langle\overline{k}_{AE},A\rangle\}_{k_E},\{\langle\overline{k}'_{AE},A\rangle\}_{k'_E}),\\
\ (\lambda\{\langle y_0,z,x_0\rangle\}_{k_A}.\mathtt{assert}(y_0=\overline{k}_{AE});\mathtt{assert}(x_0=E);\{z\}_{k_E},\ \lambda\{\langle y_0,z,x_0\rangle\}_{k'_A}.\mathtt{assert}(y_0=\overline{k}'_{AE});\mathtt{assert}(x_0=E);\{z\}_{k'_E}),\\
\ (\langle\overline{k}_{AE},A\rangle,\langle\overline{k}'_{AE},A\rangle),\\
\ (\overline{k}_{AE},\overline{k}'_{AE}),\\
\ (\langle\{\langle\overline{w},\overline{k}_B,B\rangle\}_{k_A},\lambda\{z_0\}_{k_B}.\mathtt{assert}(z_0=\overline{k}_B);\{i\}_{\overline{k}_B}\rangle,\ \langle\{\langle\overline{w}',\overline{k}'_B,B\rangle\}_{k'_A},\lambda\{z_0\}_{k'_B}.\mathtt{assert}(z_0=\overline{k}'_B);\{j\}_{\overline{k}'_B}\rangle),\\
\ (\{\langle\overline{w},\overline{k}_B,B\rangle\}_{k_A},\{\langle\overline{w}',\overline{k}'_B,B\rangle\}_{k'_A}),\\
\ (\lambda\{z_0\}_{k_B}.\mathtt{assert}(z_0=\overline{k}_B);\{i\}_{\overline{k}_B},\ \lambda\{z_0\}_{k'_B}.\mathtt{assert}(z_0=\overline{k}'_B);\{j\}_{\overline{k}'_B}),\\
\ (\{\overline{w}\}_{k_E},\{\overline{w}'\}_{k'_E})\}
\end{array}
$$

Outline

1. Logical relations for type abstraction
   [Reynolds 83] [Mitchell 91]
2. Bisimulations for type abstraction
   [Sumii-Pierce, POPL'05]
3. Bisimulations for encryption
   [Sumii-Pierce, POPL'04 & TCS]

Related Work (1/5): Logical Relations

- Semantic logical relations
  [Tait 67, Plotkin 73, Reynolds 83, Mitchell 93, etc.]
  - Suffers from the complexity and imprecision of denotational semantics for recursion
- Syntactic logical relations
  [Pitts 98, Birkedal-Harper 97, etc.]
  - Still suffers from complications for recursive functions/types

Related Work (2/5): Applicative Bisimulations

- For untyped l-calculus [Abramsky 90]
- For object calculi with universal and subtyping polymorphism [Gordon-Rees]

None can deal with type abstraction (i.e., existential polymorphism)

Penn University of Pennsylvania

東京大学 THE UNIVERSITY OF TOKYO

Related Work (3/5): Bisimulations for p-Calculi

- **For polymorphic p-calculus**
  [Pierce-Sangiorgi 97, Berger-Honda-Yoshida 03]
- **For spi-calculus**
  [Abadi-Gordon 98, Boreale-DeNicola-Pugliese 99, Borgstrom-Nestmann 02, Abadi-Fournet 01, etc.]

Incomplete, or completeness claimed but proof unpublished (or found wrong)

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Related Work (4/5): Operational Models of Types

- Indexed models [Appel et al.]
- Operational ideal models [Voillon-Melliès 04, etc.]

Only unary case (safety) considered;
Binary case (equivalence) left open

- Non-trivial in the presence of existential types

Related Work (5/5)

- **Categorical reformulation of our logical relations for perfect encryption**
  [Goubault-Larrecq-Lasota-Nowak-Zhang, CSL 04]

Future Directions (1/3)

- Translations between different forms of information hiding
  - E.g. from type abstraction to encryption
  - Challenge: How to prove full abstraction (preservation of equivalence)?
- Extension to even more expressive languages (e.g. higher-order p-calculus)

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Future Directions (2/3)

- Adoptation/generalization for other forms of information hiding (e.g. security typing)
  - Generative security levels?
- A system allowing unchecked, dynamically checked, and statically typed code <u>without losing abstraction</u>
  - C, Perl, ML, etc. coexist in peace?

Future Directions (3/3)