Theories of Information Hiding in Lambda-Calculus

Logical Relations and Bisimulations for Encryption and Type Abstraction

Eijiro Sumii

University of Pennsylvania

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,θ1),(r2,θ2)) =
    (r1*r2,θ1+θ2)
  fun re(r,θ) = r*cos(θ)
```

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

The Abstraction Property

Contextual equivalence:

$$\vdash \texttt{Cartesian} \equiv \texttt{Polar} : \texttt{Complex}$$

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

How to Prove it?

Logical relations [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 attackers must be expressed in the language (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 bisimulations in place of logical relations
- Use encryption instead of type abstraction

Penn
University of Pennsylvania

東京大学
The University of Tokyo

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]

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`
- `λx.x+1+2` and `λx.x+3` are related at type `Int→Int`
- `λx.(123,x+1+2)` and `λx.(123,x+3)` are related at type `Int→(Int×Int)`

Logical Relations for Type Abstraction

In addition to the previous cases:

- Abstract data of type α are related iff they are related by $\varphi(\alpha)$
  - where φ is a relation environment mapping abstract types to the relation between their implementations

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Example

Take

φ(`Complex.t`) =

{(`(x,y)`,`(r,θ)`) | x = r cos θ, y = r sin θ}

Then

$\varphi \vdash$ `Cartesian` ~ `Polar` : `Complex`

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

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

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Shortcoming of Logical Relations

Don't "scale" to more expressive languages

- Recursive functions (or while-loops) complicate the fundamental property
- Recursive types complicate the definition of logical relations

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

Penn
University of Pennsylvania

東京大学
The University of Tokyo

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
- {(`λx.x+1+2`, `λx.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

Bisimulations for Type Abstraction: First Try

Bisimilar functions return bisimilar results when applied to bisimilar arguments

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

THIS IS UNSOUND!

- **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 not contextually equivalent at type $\alpha \times (\alpha \times \alpha \to \texttt{int})$

But

$\{(\texttt{(1,fst)}, \texttt{(2,fst)}, \alpha \times (\alpha \times \alpha \to \texttt{int})),$
$(\texttt{1}, \texttt{2}, \alpha),$
$(\texttt{fst}, \texttt{fst}, \alpha \times \alpha \to \texttt{int})\}$

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

$$
\begin{aligned}
R = \{\ & (\texttt{Cartesian}, \texttt{Polar}, \texttt{Complex}),\\
& (\texttt{Cartesian.make}, \texttt{Polar.make},\\
& \ \texttt{real} \times \texttt{real} \rightarrow \texttt{Complex.t}),\\
& (\texttt{Cartesian.mul}, \texttt{Polar.mul},\\
& \ \texttt{Complex.t} \times \texttt{Complex.t} \rightarrow \texttt{Complex.t}),\\
& (\texttt{Cartesian.re}, \texttt{Polar.re},\\
& \ \texttt{Complex.t} \rightarrow \texttt{real}),\\
& (\texttt{(x,y)}, \texttt{(r,}\theta\texttt{)}, \texttt{Complex.t}),\\
& (\texttt{z}, \texttt{z}, \texttt{real}) \mid \texttt{x} = \texttt{r}\cos\theta,\ \texttt{y} = \texttt{r}\sin\theta\ \}
\end{aligned}
$$

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 $R \cup R^{-1}$ of the previous R and its inverse R^{-1}**

  $\Rightarrow$ 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
- { $R \cup R^{-1}$ } is <u>not</u> a bisimulation

Penn
University of Pennsylvania
東京大学
THE UNIVERSITY OF TOKYO

Soundness and Completeness

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

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**

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

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,θ_1) = decrypt_k'(c_1) in
   let (r_2,θ_2) = decrypt_k'(c_2) in
   encrypt_k'(r_1*r_2,θ_1+θ_2)
  fun re(c) =
   let (r,θ) = decrypt_k'(c) in r*cos(θ)
```

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

The Abstraction Property

Untyped contextual equivalence:

`Cartesian` ≡ `Polar`

- **Abstraction holds against any context in the language, even if untyped**

Penn
University of Pennsylvania
東京大学
THE UNIVERSITY OF TOKYO

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 $k_1 = k_2 \Leftrightarrow k_1' = k_2'$
- For any bisimilar ciphertexts $encrypt_k(v)$ and $encrypt_{k'}(v')$,
  - Neither k nor k' is in the relation, or
  - v and v' are bisimilar

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Example

$$
\begin{aligned}
R = \{\ & (\texttt{Cartesian}, \texttt{Polar}),\\
& (\texttt{Cartesian.make}, \texttt{Polar.make}),\\
& (\texttt{Cartesian.mul}, \texttt{Polar.mul}),\\
& (\texttt{Cartesian.re}, \texttt{Polar.re}),\\
& (\texttt{encrypt}_k(x,y), \texttt{encrypt}_{k'}(r,\theta)),\\
& (z, z)\ |\\
& x = r\cos\theta,\ \ y = r\sin\theta,\ \ z : \texttt{real}\ \}
\end{aligned}
$$

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 **let** $\{x\}_{e_1} = e_2$ **in** e_3 **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 $(s)e \Downarrow (t)v$ from terms e to values v
  - Annotated with the set of keys s and t before and after the evaluation

$$\frac{k \notin s \qquad (s \cup \{k\})\ [k/x]e \Downarrow (t)v}{(s)\ \text{new } x \text{ in } e \Downarrow (t)v}$$

Operational Semantics of the Language (2/2)

- **Success of decryption:**

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

- **Failure of decryption:**

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

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Other Examples

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

Penn
University of Pennsylvania

東京大学
The University of Tokyo

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, contexts play the role of the network, scheduler, and attackers by applying the receivers to the senders

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Example

1. $A \rightarrow B : \{N\}_K$
2. $B \rightarrow A : N \bmod 2$

Sys_N = `new x in`

`(encrypt`$_x$`(N),`

λ`y.(decrypt`$_x$`(y) mod 2))`

Penn
University of Pennsylvania
東京大学
THE UNIVERSITY OF TOKYO

Non-Interference: Secrecy as Equivalence

With our bisimulations, it is easy to prove

$$Sys_M \equiv Sys_N$$

for any M and N with

$$M \bmod 2 = N \bmod 2$$

Which means:

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

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Example

$$R = \{ (Sys_M, Sys_N),$$
$$(encrypt_k(M), encrypt_{k'}(N)),$$
$$(\lambda y.(decrypt_k(y) \bmod 2),$$
$$\lambda y.(decrypt_{k'}(y) \bmod 2)),$$
$$(M \bmod 2, N \bmod 2) \}$$

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 λ-calculus via interleaving

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

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Needham-Schroeder-Lowe Protocol

1. $\mathsf{B} \rightarrow \mathsf{A} : B$
2. $\mathsf{A} \rightarrow \mathsf{B} : \{N_A, A\}_{k_B}$
3. $\mathsf{B} \rightarrow \mathsf{A} : \{N_A, N_B, B\}_{k_A}$
4. $\mathsf{A} \rightarrow \mathsf{B} : \{N_B\}_{k_B}$
5. $\mathsf{B} \rightarrow \mathsf{A} : \{i\}_{N_B}$

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Encoding of Needham-Schroeder-Lowe Protocol

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

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'_B},\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_B},\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'_B}\rangle),\\
(\{\{\langle\overline{k}_{AE},A\rangle\}_{k_B},\{\{\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_B},\{\overline{w}'\}_{k'_B})\}
\end{array}
$$

Penn
University of Pennsylvania

東京大学
The University of Tokyo

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

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**

Penn
University of Pennsylvania

東京大学
The University of Tokyo

Related Work (2/5): Applicative Bisimulations

- For untyped λ-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 π-Calculi

- For polymorphic π-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**

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

Related Work (5/5)

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

Penn
University of Pennsylvania

東京大学
THE UNIVERSITY OF TOKYO

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 π-calculus)

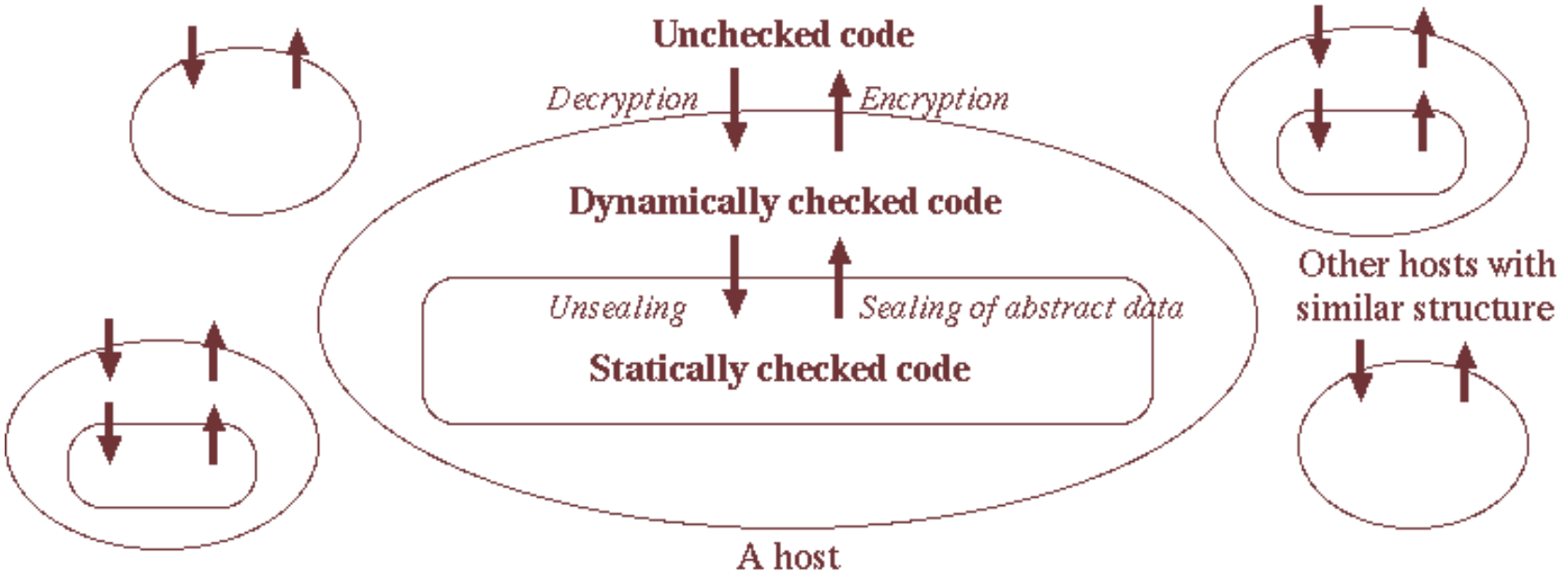
Penn
University of Pennsylvania

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 without losing abstraction
  - C, Perl, ML, etc. coexist in peace?

Future Directions (3/3)



Penn
University of Pennsylvania

東京大学
The University of Tokyo