PROVING NONINTERFERENCE BY A FULLY COMPLETE TRANSLATION TO THE SIMPLY TYPED λ-CALCULUS *

NAOKATA SHIKUMA AND ATSUSHI IGARASHI

Graduate School of Informatics, Kyoto University, Kyoto 606-8501 Japan
e-mail address: {naokata,igarashi}@kuis.kyoto-u.ac.jp

ABSTRACT. Tse and Zdancewic have formalized the notion of noninterference for Abadi et al.'s DCC in terms of logical relations and given a proof of noninterference by reduction to parametricity of System F. Unfortunately, their proof contains errors in a key lemma that their translation from DCC to System F preserves the logical relations defined for both calculi. In fact, we have found a counterexample for it. In this article, instead of DCC, we prove noninterference for *sealing calculus*, a new variant of DCC, by reduction to the basic lemma of a logical relation for the simply typed λ-calculus, using a *fully complete* translation to the simply typed λ-calculus. Full completeness plays an important role in showing preservation of the two logical relations through the translation. Also, we investigate relationship among sealing calculus, DCC, and an extension of DCC by Tse and Zdancewic and show that the first and the last of the three are equivalent.

1. INTRODUCTION

Background. Dependency analysis is a family of static program analyses to trace dependencies between inputs and outputs of a given program. For example, information flow analysis [3], binding-time analysis [8], and call tracking [20] are its instances. One of the most important correctness criteria of the dependency analysis is called *noninterference* [5], which roughly means that, for any pair of program inputs that are equivalent from the viewpoint of an observer at some dependency level (e.g., security level, binding-time), the outputs are also equivalent for the observer. Various techniques for type-based dependency analyses have been proposed, especially, in the context of language-based security [18].

Abadi et al. proposed a unifying framework called *dependency core calculus* (DCC) [1] for type-based dependency analyses for higher-order functional languages, and gave it a denotational model whose idea comes from *parametricity* [17, 24] of System F [16, 4] through other information flow analyses [7, 11]. They showed noninterference for several type systems of concrete dependency analyses by embedding them into DCC.

Recently, Tse and Zdancewic [21, 22, 23] studied the relationship between DCC and System F. First, they formalized the noninterference property for recursion-free DCC by using a syntactic *logical relation* [9]—a family of type-indexed relations, defined by induction on types, over programs—as the equivalence relations for inputs and outputs, thereby generalizing the notion of noninterference to higher-order inputs and outputs. Then, they gave a proof of noninterference by reducing it to the parametricity theorem, which was also formalized in terms of syntactic logical relations, of System F. Their technical development is summarized as follows:

(1) Define a translation $\mathcal{F}$ from DCC to System F;
(2) Prove, by induction on the structure of types, that the translation is both sound and complete—that is, it preserves the logical relations in the sense that

$$e_1 \approx_D e_2 : t \iff \mathcal{F}(e_1) \approx_F \mathcal{F}(e_2) : \mathcal{F}(t)$$

where t is a DCC type, and $\approx_D$ and $\approx_F$ represent the logical relations for DCC and System F, respectively; and
(3) Prove noninterference by reduction to the parametricity theorem of System F, using the sound and complete translation above.

Unfortunately, in the second step, their proof [21, 22, 23] contains an error[1], which we will briefly explain here. Note first that, for function types $t_1 \to t_2$, the logical relations are defined by: $e_1 \approx_x e_2 : t_1 \to t_2$ if and only if $e_1 e_1' \approx_x e_2 e_2' : t_2$ for any $e_1' \approx_x e_2' : t_1$ (x stands for either D or F) and that the type translation is homomorphic for function types, namely $\mathcal{F}(t_1 \to t_2) = \mathcal{F}(t_1) \to \mathcal{F}(t_2)$. Then, consider the case where t is a function type $t_1 \to t_2$. To show the left-to-right direction, we must show that $\mathcal{F}(e_1) M_1 \approx_F \mathcal{F}(e_2) M_2 : \mathcal{F}(t_2)$ for any $M_1 \approx_F M_2 : \mathcal{F}(t_1)$, from the assumption $e_1 \approx_D e_2 : t_1 \to t_2$, but we get stuck because there is no applicable induction hypothesis. If there existed a DCC term e such that $\mathcal{F}(e) = M$ for any System F term M of type $\mathcal{F}(t)$—in this case, we say a translation is *full* [6]— then M_1 and M_2 would be of the forms $\mathcal{F}(e_1')$ and $\mathcal{F}(e_2')$, making it possible to apply an induction hypothesis, and the whole proof would go through. Their translation, however, turns out *not* to be full; we have actually found a counterexample for the preservation of the equivalence from the failure of the fullness (see Section 6 for more details). So, although interesting, this indirect proof method fails at least for the combination of DCC and System F. Note that the noninterference property itself could be proved directly by induction on DCC typing.

Our Contributions. In this paper, we prove noninterference by Tse and Zdancewic's method in a slightly different setting: In order to obtain a fully complete translation, we change the source language to a richer one, what we call Sealing Calculus ($\lambda^{[]}$), and use a simpler target language, namely the simply typed λ-calculus $\lambda^{\to}$. Then, the basic lemma for logical relations of $\lambda^{\to}$ is used in place of the parametricity theorem.

$\lambda^{[]}$ is a simply typed λ-calculus with the notion of sealing and a simplification of a security calculus which Tse and Zdancewic proposed as an extension of DCC (we call it $\mathrm{DCC_{pc}}$ throughout this paper) [21, 22, 23]. A $\lambda^{[]}$ term $[e]_\ell$ stands for sealing e with a level ℓ, which is a degree of confidentiality of the sealed data. The sealed data can be extracted by unsealing e^ℓ. For example, let v a sealed boolean value, then $([v]_\ell)^\ell$ is evaluated to v. We control unsealing operations by a type system so that only users with relevant authority

---

can unseal. In the type system, e.g., we assign a sealing type $[\texttt{bool}]_\ell$ to $[v]_\ell$ for any user, but, $([v]_\ell)^\ell$ has type $\texttt{bool}$ only for authorized users. To take such a notion of "authorized users" into account, a type judgment is augmented with information about authority.

Then, we define a translation of $\lambda^{[]}$ to $\lambda^\to$ in the same way as Tse–Zdancewic's translation of DCC [21, 22, 23]: we encode $[v]_\ell$ and its type $[\texttt{bool}]_\ell$ by λ-abstraction $\lambda k\!:\!\alpha_\ell.\,v$ and function type $\alpha_\ell \to \texttt{bool}$, respectively, where α_ℓ is a type variable. Intuitively, a term K of type α_ℓ, if exists, will be a key of unsealing, that is, we can apply $\lambda k\!:\!\alpha_\ell.\,v$ to K and get the sealed value v. The existence of such a typable term K of α_ℓ in $\lambda^\to$ corresponds to a user's authority to unseal with ℓ in $\lambda^{[]}$. Our translation is full and, hence, there is no problem to prove noninterference property of $\lambda^{[]}$ under Tse–Zdancewic's scenario described above.

Our main technical contributions can be summarized as follows:

- Development of a sound and fully complete translation from $\lambda^{[]}$ to $\lambda^\to$;
- A proof of the noninterference theorem of $\lambda^{[]}$ by reduction to the basic lemma of $\lambda^\to$; and
- A proof of equivalence between $\lambda^{[]}$ and $\mathrm{DCC}_{\mathrm{pc}}$.

As for DCC, noninterference can be proved directly by straightforward induction in a manner quite similar to the basic lemma of $\lambda^\to$. So, the main interest would not be in the noninterference property itself but, rather, in how semantics of different calculi can be related with each other by translation. The existence of a fully complete translation means that $\lambda^{[]}$ provides syntax rich enough to express every denotation in the model (that is, $\lambda^\to$). The translation is also *fully abstract*, as our logical relation for $\lambda^{[]}$ coincides with its contextual equivalence. Also, comparing Tse–Zdancewic's translation of DCC with ours, we have found and show that, in spite of simplification, $\lambda^{[]}$ is actually equivalent to $\mathrm{DCC}_{\mathrm{pc}}$ mentioned above. This result indicates that both calculi are really improvements over DCC.

This article is an extended version of our previous paper [19]. In addition to giving detailed proofs, we have extended the earlier version of $\lambda^{[]}$ by introducing ordering on levels, as DCC or $\mathrm{DCC}_{\mathrm{pc}}$, making it easier to compare $\lambda^{[]}$ with them.

Structure of the Paper. The rest of the paper is organized as follows. Section 2 introduces $\lambda^{[]}$ with its syntax, type system, reduction, and logical relations and then the statement of the noninterference theorem. In Sections 3 and 4 we introduce $\lambda^\to$ and define a translation from $\lambda^{[]}$ to $\lambda^\to$ and its inverse. In Section 5, we complete our proof of noninterference by reducing it to the basic lemma of logical relations for $\lambda^\to$. Section 6 explains why Tse and Zdancewic's translation from DCC to System F is neither full nor sound, introduces their extension $\mathrm{DCC}_{\mathrm{pc}}$, which recovers fullness, and shows that $\lambda^{[]}$ and $\mathrm{DCC}_{\mathrm{pc}}$ are equivalent. Finally, Section 7 gives concluding remarks.

2. Sealing Calculus

In this section, we define $\lambda^{[]}$, which is the simply typed λ-calculus with sealing.

First, we will introduce two kinds of levels: *data levels* and *observer levels*. Intuitively, a data level represents a degree of confidentiality of data, while an observer level represents a capability of an observer (e.g., a user or a process) to access data. The observer can access only data whose data level ℓ is *lower than* (i.e., inside of the range of) his or her observer level π. Moreover, he or she can just obtain information depending on such data.

Then, we will define the terms, type systems, and reduction semantics of $\lambda^{[]}$ and show some basic properties. As mentioned in the previous section, we write $[e]_\ell$ for sealing a

$\lambda^{[]}$ term e with a data level ℓ. The sealed value can be extracted by unsealing e^{ℓ}, whose result must not be leaked to any observer whose observer level is not higher than ℓ. We control such dependency by the type system. In this system, information on the data level ℓ used for sealing is attached to types of sealing $[t]_{\ell}$; furthermore, type judgments, written $\Gamma \, ; \pi \, \vdash \, e \, : \, t$, are augmented by an observer level π, which is also called a *protection context* elsewhere [22, 23, 21], as well as by a typing context Γ, which is a (finite) mapping from variables to types. This judgment means that the value of e has type t as usual and, moreover, can be leaked to (any observer at) an observer level higher than π.

Finally, we will formalize equivalences for $\lambda^{[]}$ and give the formal statement of noninterference. The equivalences are indexed by observer levels. In the definition, any two values sealed at the same data level will always be considered equal, or indistinguishable, unless the observer level is higher than the data level; and then the noninterference amounts to saying that, given inputs equal at a given observer level, a typable program yields equal outputs (at the same level). So, in other words, an observer level reflects how much power one has to distinguish the extensional behavior of programs by investigating the contents of (sealed) values returned by the programs.

2.1. **Syntax.** Let $(\mathcal{L}, \sqsubseteq)$ be a poset where $\mathcal{L}$ is a finite set of data levels, ranged over by ℓ, and $\sqsubseteq$ is a partial order over $\mathcal{L}$. The metavariable π ranges over observer levels, which are finite subsets of data levels. We will often omit the qualifications "data" and "observer" for levels unless there is no confusion. Observer levels are pre-ordered as follows: $\pi_1 \sqsubseteq \pi_2$ if and only if, for any $\ell_1 \in \pi_1$, there exists $\ell_2 \in \pi_2$ such that $\ell_1 \sqsubseteq \ell_2$. We also abbreviate $\{\ell\} \sqsubseteq \pi$ to $\ell \sqsubseteq \pi$.

Remark 2.1. The notions of authorities and levels in the early version of this article [19] correspond to those of data and observer levels here. A main difference is that authorities were not given an order but data levels are partially ordered as in DCC. We have changed them to follow the standard terminology but also introduce an explicit distinction between two kinds of levels—those of data and those of observers.

Remark 2.2. We could unify data and observer levels and use a lattice, which is more standard in security calculi [1, 7], to define $\lambda^{[]}$, just as in (precisely speaking, an earlier version [22, 23] of) Tse and Zdancewic's extension of DCC. Nevertheless, we adopt a poset for data levels and the pre-ordered set induced from it for observer levels, because it would be rather complicated (and also tedious) to translate such a variant into $\lambda^{\rightarrow}$. Note that the observer levels can be viewed as a lattice by identifying any two elements that are greater than each other.

Then, the types of $\lambda^{[]}$ are defined as follows.

Definition 2.3 (Types)**.** The set of *types*, ranged over by t, t', t_1, t_2, $\ldots$, is defined as follows:
$$t ::= unit \mid t \rightarrow t \mid t \times t \mid t + t \mid [t]_{\ell}$$
We call $[t]_{\ell}$ a *sealing type*.

We define the terms of $\lambda^{[]}$ below. The metavariables x, y, and z (possibly with subscripts) range over the denumerable set of *variables*.

Definition 2.4 (Terms)**.** The set of *terms*, ranged over by e, e', e_1, e_2, $\ldots$, is defined as follows:

$$e ::= x \mid () \mid \lambda x{:}t.\, e \mid e\, e \mid \langle e,\, e \rangle \mid \pi_1(e) \mid \pi_2(e) \mid \iota_1(e) \mid \iota_2(e)$$
$$\mid (\mathbf{case}\, e\, \mathbf{of}\, \iota_1(x_1).e \mid \iota_2(x_2).e) \mid [e]_\ell \mid e^\ell$$

Terms of $\lambda^{[]}$ include variable, the unit value, λ-abstraction, application, pairing, projection, injection, and case analysis. As usual, x is bound in e of $\lambda x{:}t.\, e$ and x_1 and x_2 are bound in e_1 and e_2 of $(\mathbf{case}\, e_0\, \mathbf{of}\, \iota_1(x_1).e_1 \mid \iota_2(x_2).e_2)$, respectively. We say, for $[e]_\ell$, e is *sealed at ℓ*, and call $[e]_\ell$ and e^ℓ a *sealing term* and an *unsealing term*, respectively. In this paper, α-conversions are defined in a customary manner and implicit α-conversions are assumed to make all the bound variables distinct from other (bound and free) variables.

2.2. Type System.

As mentioned above, the form of type judgment of $\lambda^{[]}$ is $\Gamma\,;\, \pi \vdash e : t$. This judgment is read as "e is given type t at observer level π under context Γ." The intuition is that the computation of e depends on only data levels lower than π, and so the information on its value can be leaked only to an observer level π', which is higher than π.

The typing rules of $\lambda^{[]}$ are given as follows:

$$\frac{x : t \in \Gamma}{\Gamma\,;\, \pi \vdash x : t} \tag{ST-Var}$$

$$\frac{\Gamma,\, x : t_1\,;\, \pi \vdash e : t_2}{\Gamma\,;\, \pi \vdash \lambda x{:}t_1.\, e : t_1 \rightarrow t_2} \tag{ST-Abs}$$

$$\frac{\Gamma\,;\, \pi \vdash e : t_1 \rightarrow t_2 \qquad \Gamma\,;\, \pi \vdash e' : t_1}{\Gamma\,;\, \pi \vdash e\, e' : t_2} \tag{ST-App}$$

$$\frac{\Gamma\,;\, \pi \vdash e_1 : t_1 \qquad \Gamma\,;\, \pi \vdash e_2 : t_2}{\Gamma\,;\, \pi \vdash \langle e_1,\, e_2 \rangle : t_1 \times t_2} \tag{ST-Pair}$$

$$\frac{\Gamma\,;\, \pi \vdash e : t_1 \times t_2 \qquad i \in \{1,\, 2\}}{\Gamma\,;\, \pi \vdash \pi_i(e) : t_i} \tag{ST-Proj}$$

$$\frac{\Gamma\,;\, \pi \vdash e : t_i \qquad i \in \{1,\, 2\}}{\Gamma\,;\, \pi \vdash \iota_i(e) : t_1 + t_2} \tag{ST-Inj}$$

$$\frac{\Gamma\,;\, \pi \vdash e : t_1 + t_2 \qquad \Gamma,\, x_1 : t_1\,;\, \pi \vdash e_1 : t \qquad \Gamma,\, x_2 : t_2\,;\, \pi \vdash e_2 : t}{\Gamma\,;\, \pi \vdash (\mathbf{case}\, e\, \mathbf{of}\, \iota_1(x_1).e_1 \mid \iota_2(x_2).e_2) : t} \tag{ST-Case}$$

$$\frac{\Gamma\,;\, \pi \cup \{\ell\} \vdash e : t}{\Gamma\,;\, \pi \vdash [e]_\ell : [t]_\ell} \tag{ST-Seal}$$

$$\frac{\Gamma\,;\, \pi \vdash e : [t]_\ell \qquad \ell \sqsubseteq \pi}{\Gamma\,;\, \pi \vdash e^\ell : t} \tag{ST-Unseal}$$

All the rules but the last two are straightforward. The rule (ST-SEAL) for sealing means that, by sealing with ℓ, it is legal to leak $[e]_\ell$ to an observer level which is not higher than ℓ: at such an observer level, however, e cannot be unsealed, as is shown in the rule (ST-UNSEAL) for unsealing.

Example 2.5. The following judgment

$$\cdot\,;\,\pi \vdash \lambda x\!:\![t_1 + t_2]_{\ell_1}.\,[(\mathbf{case}\ x^{\ell_1}\ \mathbf{of}\ \iota_1(x_1).\iota_1([x_1]_{\ell_3})\,|\,\iota_2(x_2).\iota_2([x_2]_{\ell_3}))]_{\ell_2}$$
$$:\,[t_1 + t_2]_{\ell_1} \rightarrow [[t_1]_{\ell_3} + [t_2]_{\ell_3}]_{\ell_2}$$

is derivable if and only if $\ell_1 \sqsubseteq \pi \cup \{\ell_2\}$, which is required at unsealing x of $[t_1 + t_2]_{\ell_1}$ with ℓ_1—the observer level there is $\pi \cup \{\ell_2\}$ and must be higher than the data level ℓ_1.

The type constructor $[\cdot]_\ell$ is very similar to the (indexed) monadic type constructor T_ℓ in DCC [1]. In fact, the logical relations we will define for $\lambda^{[]}$ are essentially the same as those defined for DCC and a main idea of the translation from $\lambda^{[]}$ to $\lambda^{\rightarrow}$ is also the same as that from DCC to System F [21, 22, 23]. Nevertheless, we have chosen a different symbol as the monadic bind construct is no longer used in $\lambda^{[]}$ and, as a result, the type system is fairly different from DCC. We will give a more detailed comparison with DCC (and its extension [21, 22, 23]) in Section 6.

2.3. **Reduction.** The *reduction relation* for $\lambda^{[]}$ is written $e \longrightarrow e'$, which expresses that e is *reduced* to e' by applying one of the following rules to a subterm of e.

$$
\begin{array}{rcl}
(\lambda x\!:\!t.\,e_1)\,e_2 & \longrightarrow & [e_2/x]e_1 \\
\pi_i(\langle e_1,\,e_2 \rangle) & \longrightarrow & e_i \\
(\mathbf{case}\ \iota_i(e)\ \mathbf{of}\ \iota_1(x_1).e_1\,|\,\iota_2(x_2).e_2) & \longrightarrow & [e/x_i]e_i \\
([e]_\ell)^\ell & \longrightarrow & e
\end{array}
$$

We write $[e/x]$ for a capture-avoiding substitution of e for the free occurrences of variable x. All rules are straightforward. The last rule says that the term sealed by ℓ is opened by the same level. In what follows, we use v for *normal forms*, that is, terms which cannot be reduced anymore. Note that $\lambda x\!:\!t.\,([x]_\ell)^\ell$ is *not* a normal form, since the reduction is full, that is, even a redex under λ-abstraction can be reduced. We write $\longrightarrow^*$ for the reflexive transitive closure of $\longrightarrow$.

2.4. **Basic Properties.** We list some basic properties of $\lambda^{[]}$. The first lemma below means that, if e is well typed at some observer level, then it is also well typed at a higher level.

Lemma 2.6 (Observer Level Monotonicity)**.** *If* $\Gamma\,;\,\pi_1 \vdash e : t$ *and* $\pi_1 \sqsubseteq \pi_2$, *then* $\Gamma\,;\,\pi_2 \vdash e : t$, *and the derivations of these judgments have the same size.*

Proof. By induction on the derivation of $\Gamma\,;\,\pi_1 \vdash e : t$, using the fact that $\pi_1 \cup \pi \sqsubseteq \pi_2 \cup \pi$ if $\pi_1 \sqsubseteq \pi_2$. □

Lemma 2.7 (Substitution Property)**.** *If* $\Gamma\,;\,\pi \vdash e : t$ *and* $\Gamma, x : t\,;\,\pi \vdash e' : t'$, *then* $\Gamma\,;\,\pi \vdash [e/x]e' : t'$

Proof. By induction on the derivation of $\Gamma, x : t\,;\,\pi \vdash e' : t'$, using Lemma 2.6. □

The following three theorems are standard.

Theorem 2.8 (Subject Reduction). *If $\Gamma \,;\, \pi \,\vdash\, e : t$ and $e \longrightarrow e'$, then $\Gamma \,;\, \pi \,\vdash\, e' : t$.*

Proof. By induction on the derivation of $\Gamma \,;\, \pi \,\vdash\, e : t$, using Lemmas 2.6 and 2.7. □

Theorem 2.9 (Strong Normalization). *If $\Gamma \,;\, \pi \,\vdash\, e : t$, then e is* strongly normalizing, *that is, there is no infinite sequence of reductions which starts from e.*

Proof. Define a translation from $\lambda^{[]}$ into the simply typed λ-calculus as follows:

$$([t]_\ell)^\dagger = unit \to t^\dagger$$
$$([e]_\ell)^* = \lambda_\_ \colon unit.\, e^*$$
$$(e^\ell)^* = e^*\,().$$

This translation preserves typing and maps a reduction $e_1 \longrightarrow e_2$ to $e_1^* \longrightarrow^+ e_2^*$, where $\longrightarrow^+$ is the transitive closure of $\longrightarrow$. So, from strong normalization for the simply typed λ-calculus (see, e.g., [9]), we conclude one for $\lambda^{[]}$. □

Theorem 2.10 (Church-Rosser Property). *If $\Gamma \,;\, \pi \,\vdash\, e : t$ and $e \longrightarrow^* e_1$ and $e \longrightarrow^* e_2$, then there exists a term e' such that $e_i \longrightarrow^* e'$ $(i = 1, 2)$.*

Proof. By Theorem 2.9 and Newman's Lemma [13], it suffices to show that the reduction is weakly confluent: If $\Gamma \,;\, \pi \,\vdash\, e : t$ and $e \longrightarrow e_1$ and $e \longrightarrow e_2$, then there exists a term e' such that $e_i \longrightarrow^* e'$ $(i = 1, 2)$. This is easy. □

2.5. **Contextual Equivalence, Noninterference, and Logical Relations.** Now we formalize equivalence of terms from the viewpoint of an observer at a given level as *contextual equivalence*, and then state a formalization of noninterference.

We say that e_1 and e_2 are contextually equivalent at observer level π if $C[e_1]$ and $C[e_2]$ are evaluated to the same value for any context $C[\cdot]$ typed at π. Note that the equivalence is indexed by an observer level. We define contextual equivalence $\overset{\text{ctx}}{=}_\pi$ as follows:

Definition 2.11 (Contextual Equivalence for $\lambda^{[]}$). Assume that $\cdot \,;\, \pi \,\vdash\, e_i : t$ for $i = 1, 2$ (we write $\cdot$ for the empty variable context). The relation $e_1 \overset{\text{ctx}}{=}_\pi e_2 : t$ is defined by: $e_1 \overset{\text{ctx}}{=}_\pi e_2 : t$ if and only if $f e_1 \overset{\text{nf}}{=} f e_2$ for any f such that $\cdot \,;\, \pi \,\vdash\, f : t \to \texttt{bool}$. Here, $e \overset{\text{nf}}{=} e'$ means that e and e' have the same normal form and $\texttt{bool}$ stands for $unit + unit$.

Here we use functions as contexts without loss of generality, because, by Strong Normalization and Church-Rosser, $C[e]$ and $(\lambda x \colon t.\, C[x])\, e$ has a unique normal form, where t is the type of e.

The following proposition shows that an observer level in the contextual equivalence reflects an observer's distinguishability, in other words, that an observer at a lower level can distinguish no more terms than another at a higher.

Proposition 2.12. *Assume that $\cdot \,;\, \pi_1 \,\vdash\, e_i : t$ for $i = 1, 2$. If $\pi_1 \sqsubseteq \pi_2$ and $e_1 \overset{\text{ctx}}{=}_{\pi_2} e_2 : t$, then $e_1 \overset{\text{ctx}}{=}_{\pi_1} e_2 : t$.*

Proof. Take a function f such that $\cdot \,;\, \pi_1 \,\vdash\, f : t \to \texttt{bool}$. By Observer Level Monotonicity (Proposition 2.6), $\cdot \,;\, \pi_2 \,\vdash\, f : t \to \texttt{bool}$ and $\cdot \,;\, \pi_2 \,\vdash\, e_i : t$ $(i = 1, 2)$. By assumption, $f e_1 \overset{\text{nf}}{=} f e_2$, and so $e_1 \overset{\text{ctx}}{=}_{\pi_1} e_2 : t$. □

We use γ to represent a simultaneous substitution of terms for variables and write $\gamma_1 \stackrel{\text{ctx}}{=}_\pi$ $\gamma_2 : \Gamma$ if $dom(\gamma_1) = dom(\gamma_2) = dom(\Gamma)$ and $\gamma_1(x) \stackrel{\text{ctx}}{=}_\pi \gamma_2(x) : \Gamma(x)$ for any $x \in dom(\gamma_1)$. Then, the noninterference is defined as follows:

Definition 2.13 (Noninterference). Take e such that $\Gamma ; \pi \vdash e : t$. *The well typed term e satisfies noninterference*, if and only if, $\gamma_1(e) \stackrel{\text{ctx}}{=}_\pi \gamma_2(e) : t$ for any γ_1 and γ_2 such that $\gamma_1 \stackrel{\text{ctx}}{=}_\pi \gamma_2 : \Gamma$.

As mentioned before, noninterference means that, for any pair of program inputs that are equivalent from the viewpoint of an observer at some security level, the outputs are also equivalent for the observer. Here, substitutions γ_1 and γ_2 play roles of equivalent inputs to program e. So, this property specifies the correctness of the type system as a dependency analysis.

Though we want to show that any well typed term satisfies the noninterference above, this is hard due to the following generally-known fact: it is difficult, in general, to show given two terms are contextually equivalent. The reason is that we must take account of *all* contexts but proof by induction on the structure of contexts does not usually work.

To solve this problem, we use the well-known technique of *logical relations* [9, 14], which will be shown to be equivalent to the contextual equivalences, and state the noninterference theorem in terms of the logical relations.

As the contextual equivalence above, the logical relations (for close terms and closed normal forms) are indexed by observer levels as well as types. A judgment $e_1 \approx_\pi e_2 : t$ means that closed terms e_1 and e_2 of type t are logically related at observer level π. Similarly, $v_1 \sim_\pi v_2 : t$ means that closed normal forms v_1 and v_2 of t are logically related at π. We assume $\cdot ; \pi \vdash e_i : t$ and $\cdot ; \pi \vdash v_i : t$ for $i = 1, 2$.

Definition 2.14 (Logical Relations for $\lambda^{[]}$). The relations $v_1 \sim_\pi v_2 : t$ and $e_1 \approx_\pi e_2 : t$ are defined by the following rules:

$$() \sim_\pi () : unit \tag{SL-Unit}$$

$$\frac{\forall (e_1 \approx_\pi e_2 : t_1). \, v_1 \, e_1 \approx_\pi v_2 \, e_2 : t_2}{v_1 \sim_\pi v_2 : t_1 \to t_2} \tag{SL-Fun}$$

$$\frac{v_{11} \sim_\pi v_{21} : t_1 \qquad v_{12} \sim_\pi v_{22} : t_2}{\langle v_{11}, \, v_{12} \rangle \sim_\pi \langle v_{21}, \, v_{22} \rangle : t_1 \times t_2} \tag{SL-Pair}$$

$$\frac{v_1 \sim_\pi v_2 : t_i \qquad i \in \{1, \, 2\}}{\iota_i(v_1) \sim_\pi \iota_i(v_2) : t_1 + t_2} \tag{SL-Inj}$$

$$\frac{\ell \not\sqsubseteq \pi}{[v_1]_\ell \sim_\pi [v_2]_\ell : [t]_\ell} \tag{SL-Seal1}$$

$$\frac{v_1 \sim_\pi v_2 : t \qquad \ell \sqsubseteq \pi}{[v_1]_\ell \sim_\pi [v_2]_\ell : [t]_\ell} \tag{SL-Seal2}$$

$$\frac{e_1 \longrightarrow^* v_1 \qquad e_2 \longrightarrow^* v_2 \qquad v_1 \sim_\pi v_2 : t}{e_1 \approx_\pi e_2 : t} \tag{SL-Term}$$

Most rules are straightforward. In the rule (SL-Fun), the premise is the abbreviation of the following: $\forall e_1.\ \forall e_2.\ e_1 \approx_\pi e_2 : t_1 \Rightarrow v_1 e_1 \approx_\pi v_2 e_2 : t_2$. There are two rules for $[v_1]_\ell \sim_\pi [v_2]_\ell : [t]_\ell$. When $\ell \sqsubseteq \pi$, an observer at π can examine v_i by unsealing $[v_i]_\ell$ ($i = 1, 2$), so these sealing terms are equivalent only when its contents are equivalent. Otherwise, the observer cannot distinguish them at all and those terms are always regarded equivalent.

Example 2.15. We write true and false, respectively, for $\iota_1(())$ and $\iota_2(())$. Let L and H data levels and suppose that L is strictly lower than H. Take any e_i such that $\cdot;\ L \vdash e_i : [\text{bool}]_H$ ($i = 1, 2$). Then $e_1 \approx_L e_2 : [\text{bool}]_H$. This follows from the facts that $[c_1]_H \sim_L [c_2]_H : [\text{bool}]_H$ where $c_1, c_2 \in \{\text{true}, \text{false}\}$ and that each e_i has either normal form $[\text{true}]_H$ or $[\text{false}]_H$.

We define $\gamma_1 \approx_\pi \gamma_2 : \Gamma$ similarly to $\gamma_1 \overset{\text{ctx}}{=}_\pi \gamma_2 : \Gamma$. Then, the noninterference theorem is stated as follows:

Theorem 2.16 (Noninterference). *If $\Gamma;\ \pi \vdash e : t$ and $\gamma_1 \approx_\pi \gamma_2 : \Gamma$, then $\gamma_1(e) \approx_\pi \gamma_2(e) : t$.*

We will give a proof in Section 5.

Example 2.17. Here, we use the same notations as Example 2.15. Take a function f such that $\cdot;\ L \vdash f : [\text{bool}]_H \to [\text{bool}]_L$. Now we will show that f is a constant function. By the theorem above, $f \approx_L f : [\text{bool}]_H \to [\text{bool}]_L$. From (SL-Term), the discussion in Example 2.15 and (SL-Fun), $f e_1 \approx_L f e_2 : [\text{bool}]_L$. $f e_i$ has a normal form $[c_i]_L$ where some $c_i \in \{\text{true}, \text{false}\}$ ($i = 1, 2$) and, by (SL-Term), $[c_1]_L \sim_L [c_2]_L : [\text{bool}]_L$. So, by (SL-Seal2), $c_1 = c_2$, which means that f always returns a constant value.

Also, from the noninterference theorem (Theorem 2.16), it follows that the logical relations exactly coincide with the contextual equivalences above, and hence, in terms of the latter as well as the former, the noninterference theorem also holds.

Theorem 2.18. $e_1 \approx_\pi e_2 : t$ *if and only if* $e_1 \overset{\text{ctx}}{=}_\pi e_2 : t$.

Proof. First, we show the right from the left. Suppose that $e_1 \approx_\pi e_2 : t$. Take arbitrary f such that $\cdot;\ \pi \vdash f : t \to \text{bool}$. By Noninterference Theorem, $f \approx_\pi f : t \to \text{bool}$, and by (SL-Term) and (SL-Fun), $f e_1 \approx_\pi f e_2 : \text{bool}$. By (SL-Term), (SL-Inj) and (SL-Unit), $f e_1 \overset{\text{nf}}{=} f e_2$ and hence $e_1 \overset{\text{ctx}}{=}_\pi e_2 : t$.

Next, we prove the converse above by induction on the structure of t. Assume that $e_1 \overset{\text{ctx}}{=}_\pi e_2 : t$. We show only the main cases:

Case ($t = t_1 \to t_2$). Take arbitrary e_1' and e_2' such that $e_1' \approx_\pi e_2' : t_1$. By the left-to-right of Theorem 2.18 (which has been already shown in the first part of this proof), $e_1' \overset{\text{ctx}}{=}_\pi e_2' : t_1$. Take arbitrary f such that $\cdot;\ \pi \vdash f : t_2 \to \text{bool}$, then $f(e_1 e_1') \overset{\text{nf}}{=} f(e_1 e_2')$ because $e_1' \overset{\text{ctx}}{=}_\pi e_2' : t_1$. Also, by assumption, $f(e_1 e_2') \overset{\text{nf}}{=} f(e_2 e_2')$, and hence $f(e_1 e_1') \overset{\text{nf}}{=} f(e_2 e_2')$ by transitivity of $=_{\text{nf}}$. So, $e_1 e_1' \overset{\text{ctx}}{=}_\pi e_2 e_2' : t_2$, and by the induction hypothesis for t_2, $e_1 e_1' \approx_\pi e_2 e_2' : t_2$, therefore $e_1 \approx_\pi e_2 : t_1 \to t_2$.

Case ($t = [t_1]_\ell$). We have two subcases according to whether $\ell \sqsubseteq \pi$ or not. If $\ell \sqsubseteq \pi$, then, by Strong Normalization (Theorem 2.9), there are normal forms v_1 and v_2 such that $\cdot;\ \pi \vdash v_i : t_1$ and $e_i \longrightarrow^* [v_i]_\ell$ for $i = 1, 2$. Then, it must be the case that $v_1 \overset{\text{ctx}}{=}_\pi v_2 : t_1$.

(Otherwise, there would be a term f such that $\cdot\,; \pi \vdash f : t_1 \to \texttt{bool}$ and $fv_1 \neq_{\text{nf}} fv_2$. Let f' be $\lambda x\!:\![t_1]_\ell.\, fx^\ell$, then $\cdot\,; \pi \vdash f' : [t_1]_\ell \to \texttt{bool}$ and $f'e_1 \neq_{\text{nf}} f'e_2$, and hence, $e_1 \neq^\pi_{\text{ctx}} e_2 : [t_1]_\ell$, but this is a contradiction.) Applying the induction hypothesis for t_1, $v_1 \approx_\pi v_2 : t_1$, which is equivalent to $v_1 \sim_\pi v_2 : t_1$, so $e_1 \approx_\pi e_2 : [t_1]_\ell$. The case $\ell \not\sqsubseteq \pi$ is trivial. $\qquad\square$

3. The Simply Typed λ-calculus

We review the simply typed λ-calculus $\lambda^\to$ briefly with logical relations for it.

3.1. **Definition of $\lambda^\to$.** $\lambda^\to$ introduced here is a standard one with unit, base, function, product, and sum types. We assume that base types, written α_ℓ ($\ell \in \mathcal{L}$), have one-to-one correspondence with data levels. We use metavariables M for terms and A for types. The syntax of $\lambda^\to$ is given as follows:

$$
\begin{array}{rcl}
A & ::= & \alpha_\ell \mid \mathit{unit} \mid A \to A \mid A \times A \mid A + A \\
M & ::= & x \mid () \mid \lambda x\!:\!A.\,M \mid M\,M \mid \langle M,\,M \rangle \mid \pi_i(M) \mid \iota_i(M) \\
  &   & \mid (\textbf{case } M \textbf{ of } \iota_1(x_1).M \mid \iota_2(x_2).M)
\end{array}
$$

Note that base type α_ℓ has neither constants nor closed terms. The reason is that, as mentioned in Section 1, we will use a term of type α_ℓ as a key for opening a sealing at level ℓ and such a key should be permitted only to privileged users. See Section 4 for details.

The form of type judgment of $\lambda^\to$ is $\Delta \vdash M : A$, where Δ is a (finite) mapping from variables to $\lambda^\to$ types. The typing rules are given as follows:

$$
\frac{x : A \in \Gamma}{\Delta \vdash x : A} \tag{LT-Var}
$$

$$
\Delta \vdash () : \mathit{unit} \tag{LT-Unit}
$$

$$
\frac{\Delta,\, x : A \vdash M : B}{\Delta \vdash \lambda x\!:\!A.\,M : A \to B} \tag{LT-Abs}
$$

$$
\frac{\Delta \vdash M : A \to B \qquad \Delta \vdash N : A}{\Delta \vdash M\,N : B} \tag{LT-App}
$$

$$
\frac{\Delta \vdash M : A \qquad \Delta \vdash N : B}{\Delta \vdash \langle M,\, N \rangle : A \times B} \tag{LT-Pair}
$$

$$
\frac{\Delta \vdash M : A_1 \times A_2 \qquad i \in \{1,\,2\}}{\Delta \vdash \pi_i(M) : A_i} \tag{LT-Proj}
$$

$$
\frac{\Delta \vdash M : A_i \qquad i \in \{1,\,2\}}{\Delta \vdash \iota_i(M) : A_1 + A_2} \tag{LT-Inj}
$$

$$
\frac{\Delta \vdash M : A_1 + A_2 \qquad \Delta,\, x_1 : A_1 \vdash N_1 : B \qquad \Delta,\, x_2 : A_2 \vdash N_2 : B}{\Delta \vdash (\textbf{case } M \textbf{ of } \iota_1(x_1).N_1 \mid \iota_2(x_2).N_2) : B} \tag{LT-Case}
$$

The reduction of $\lambda^{\rightarrow}$ terms consists of standard β-reduction

$$
\begin{aligned}
(\lambda x\!:\!A.\,M_1)\,M_2 &\longrightarrow [M_2/x]M_1 \\
\pi_i(\langle M_1,\,M_2\rangle) &\longrightarrow M_i \\
(\mathbf{case}\ \iota_i(M)\ \mathbf{of}\ \iota_1(x_1).M_1\,|\,\iota_2(x_2).M_2) &\longrightarrow [M/x_i]M_i
\end{aligned}
$$

and the following commutative conversion.

$$
\frac{(x_1, x_2 \notin FV(M'))}{(\mathbf{case}\ M\ \mathbf{of}\ \iota_1(x_1).M_1\,|\,\iota_2(x_2).M_2)\,M' \longrightarrow \mathbf{case}\ M\ \mathbf{of}\ \iota_1(x_1).M_1\,M'\,|\,\iota_2(x_2).M_2\,M'}
$$

$$
\frac{(i \in \{1,2\})}{\pi_i(\mathbf{case}\ M\ \mathbf{of}\ \iota_1(x_1).M_1\,|\,\iota_2(x_2).M_2) \longrightarrow \mathbf{case}\ M\ \mathbf{of}\ \iota_1(x_1).\pi_i(M_1)\,|\,\iota_2(x_2).\pi_i(M_2)}
$$

$$
\frac{(x_1, x_2 \notin FV(M_1') \cup FV(M_2'))}{\begin{aligned}\mathbf{case}\ (\mathbf{case}\ &M\ \mathbf{of}\ \iota_1(x_1).M_1\,|\,\iota_2(x_2).M_2\ \mathbf{of}\ \iota_1(y_1).M_1'\,|\,\iota_2(y_2).M_2' \\ \longrightarrow\ \ &\mathbf{case}\ M\ \mathbf{of}\ \iota_1(x_1).(\mathbf{case}\ M_1\ \mathbf{of}\ \iota_1(y_1).M_1'\,|\,\iota_2(y_2).M_2') \\ |\ \ &\iota_2(x_2).(\mathbf{case}\ M_2\ \mathbf{of}\ \iota_1(y_1).M_1'\,|\,\iota_2(y_2).M_2')\end{aligned}}
$$

As in $\lambda^{[\,]}$, the reduction for $\lambda^{\rightarrow}$ is full, too. Here, we write $FV(M)$ for the set of free variables in M. In what follows, we use V for normal forms. For example, by the first and second commutative conversion rules,

$$
\begin{aligned}
&\lambda z\!:\!unit + unit.\,\pi_i((\mathbf{case}\ z\ \mathbf{of}\ \iota_1(x_1).y_1\,|\,\iota_2(x_2).y_2)z) \\
&\longrightarrow \lambda z\!:\!unit + unit.\,\pi_i((\mathbf{case}\ z\ \mathbf{of}\ \iota_1(x_1).y_1\,z\,|\,\iota_2(x_2).y_2\,z)) \\
&\longrightarrow \lambda z\!:\!unit + unit.\,(\mathbf{case}\ z\ \mathbf{of}\ \iota_1(x_1).\pi_i(y_1\,z)\,|\,\iota_2(x_2).\pi_i(y_2\,z)),
\end{aligned}
$$

which is a normal form.

The resulting calculus (with commutative conversion) satisfies the standard properties of subject reduction, Church-Rosser, and strong normalization [2]. We say (the type derivation $\Delta \vdash M : A$ of) a term satisfies the subformula property when any type in the derivation is a subexpression of either A or a type occurring in Δ. Then, any well typed term can reduce to the one that satisfies the subformula property as in the theorem below, which makes it easy to ensure the fullness of the translation.

Theorem 3.1 (Subformula Property). *If $\Delta \vdash M : A$, then there exists a normal form V such that $M \longrightarrow^* V$ and $\Delta \vdash V : A$, which satisfies the subformula property. Also, all the subderivations satisfy the subformula property.*

Remark 3.2. Commutative conversion is necessary for the above theorem to hold. Without commutative conversion,

$$
\lambda x\!:\!unit + unit.\,((\mathbf{case}\ x\ \mathbf{of}\ \iota_1(x_1).\lambda y\!:\!unit.\,()\,|\,\iota_2(x_2).\lambda y\!:\!unit.\,()))\,()
$$

of type $unit + unit \rightarrow unit$ would be a normal form, which does not satisfy the subformula property, because a subterm $\lambda y\!:\!unit.\,()$ has type $unit \rightarrow unit$, which does not occur in $unit + unit \rightarrow unit$. This theorem also requires full reduction, which allows any redex (even under λ) to reduce.

As mentioned above, we will view terms of type α_ℓ as keys. What really matters in the development below is *whether* any key of a given type exists or not and it is is not significant what kind of keys exist. Thus we identify all keys by introducing a (typed) equivalence relation $\Delta \vdash M_1 \equiv M_2 : A$.

Definition 3.3. The relation $\Delta \vdash M_1 \equiv M_2 : A$ is defined as the least relation closed under the rules below:

$$\frac{\Delta \vdash M_1 : \alpha_\ell \qquad \Delta \vdash M_2 : \alpha_\ell}{\Delta \vdash M_1 \equiv M_2 : \alpha_\ell} \tag{A-Key}$$

$$\Delta, x : A \vdash x \equiv x : A \tag{A-Var}$$

$$\Delta \vdash () \equiv () : \mathit{unit} \tag{A-Unit}$$

$$\frac{\Delta, x : A_1 \vdash M \equiv M' : A_2}{\Delta \vdash \lambda x{:}A_1.\, M \equiv \lambda x{:}A_1.\, M' : A_1 \rightarrow A_2} \tag{A-Abs}$$

$$\frac{\Delta \vdash M_1 \equiv M_1' : A_1 \rightarrow A_2 \qquad \Delta \vdash M_2 \equiv M_2' : A_1}{\Delta \vdash M_1\, M_2 \equiv M_1'\, M_2' : A_2} \tag{A-App}$$

$$\frac{\Delta \vdash M_1 \equiv M_1' : A_1 \qquad \Delta \vdash M_2 \equiv M_2' : A_2}{\Delta \vdash \langle M_1,\, M_2 \rangle \equiv \langle M_1',\, M_2' \rangle : A_1 \times A_2} \tag{A-Pair}$$

$$\frac{\Delta \vdash M \equiv M' : A_1 \times A_2 \qquad i \in \{1, 2\}}{\Delta \vdash \pi_i(M) \equiv \pi_i(M') : A_i} \tag{A-Proj}$$

$$\frac{\Delta \vdash M \equiv M' : A_i \qquad i \in \{1, 2\}}{\Delta \vdash \iota_i(M) \equiv \iota_i(M') : A_1 + A_2} \tag{A-Inj}$$

$$\frac{\Delta \vdash M \equiv M' : A_1 + A_2 \qquad \Delta, x_1 : A_1 \vdash M_1 \equiv M_1' : A \qquad \Delta, x_2 : A_2 \vdash M_2 \equiv M_2' : A}{\Delta \vdash (\mathbf{case}\ M\ \mathbf{of}\ \iota_1(x_1).M_1 \,|\, \iota_2(x_2).M_2) \equiv (\mathbf{case}\ M'\ \mathbf{of}\ \iota_1(x_1).M_1' \,|\, \iota_2(x_2).M_2') : A} \tag{A-Case}$$

The rule (A-Key) signifies that all keys are identified. Clearly, $\Delta \vdash M \equiv M : A$ is equivalent to $\Delta \vdash M : A$.

Lemma 3.4 ($\equiv$ is Equivalence)**.** *Given Δ and A, the binary relation $\Delta \vdash \cdot \equiv \cdot : A$ on terms is an equivalence relation, that is, reflexive, symmetric, and transitive.*

Proof. Easy.   □

The following lemma says that two terms which differ only in subterms of type α_ℓ are equivalent via $\equiv$.

Lemma 3.5. *Assume that $\Delta \vdash M : A$. Take an occurrence M_1 of type α_ℓ in M. Suppose that M_1 freely occurs in M, that is, no free variable of M_1 is bound in the occurrence. If $\Delta \vdash M_2 : \alpha_\ell$, then $\Delta \vdash M \equiv [M_2/M_1]M : A$, where $[M_2/M_1]M$ is a result of capture avoiding replacement of the occurrence M_1 in M by M_2. In general, this holds for simultaneous replacing too.*

Proof. By induction on the derivation of $\Delta \vdash M : A$.   □

3.2. **Logical Relations for $\lambda^\rightarrow$.** We define syntactic logical relations for $\lambda^\rightarrow$ in the standard manner. As for $\lambda^{[]}$, there are relations for (this time, possibly open) terms and normal forms, written $\Delta \vdash M_1 \approx M_2 : A$ (read "terms M_1 and M_2 of type A are logically related under context Δ") and $\Delta \vdash V_1 \sim V_2 : A$ (read similarly), respectively. We assume that $\Delta \vdash M_i : A$ and $\Delta \vdash V_i : A$ for $i = 1, 2$.

Definition 3.6 (Logical Relations for $\lambda^\rightarrow$). The relations $\Delta \vdash M_1 \approx M_2 : A$ and $\Delta \vdash V_1 \sim V_2 : A$ are the least relation closed under the following rules:

$$\Delta \vdash () \sim () : \mathit{unit} \tag{LL-Unit}$$

$$\Delta \vdash V_1 \sim V_2 : \alpha_\ell \tag{LL-KT}$$

$$\frac{\Delta \vdash V_{11} \sim V_{21} : A_1 \qquad \Delta \vdash V_{12} \sim V_{22} : A_2}{\Delta \vdash \langle V_{11}, V_{12} \rangle \sim \langle V_{21}, V_{22} \rangle : A_1 \times A_2} \tag{LL-Pair}$$

$$\frac{\Delta \vdash V_1 \sim V_2 : A_i \qquad i \in \{1, 2\}}{\Delta \vdash \iota_i(V_1) \sim \iota_i(V_2) : A_1 + A_2} \tag{LL-Inj}$$

$$\frac{\forall (\Delta \vdash M_1 \approx M_2 : A_1).\, \Delta \vdash V_1\, M_1 \approx V_2\, M_2 : A_2}{\Delta \vdash V_1 \sim V_2 : A_1 \rightarrow A_2} \tag{LL-Fun}$$

$$\frac{M_1 \longrightarrow^* V_1 \qquad M_2 \longrightarrow^* V_2 \qquad \Delta \vdash V_1 \sim V_2 : A}{\Delta \vdash M_1 \approx M_2 : A} \tag{LL-Term}$$

The rule (LL-KT) corresponds to (A-Key) and means that the number of keys to open a sealing with ℓ is at most one. Although we could give a more general definition of syntactic logical relations, where the relation for type α_ℓ is parameterized, and prove the basic lemma for them below, but, in this paper, we do not need such general settings and just take the restricted version above for simplicity.

Example 3.7. Take M_i such that $k : \alpha_\mathsf{L} \vdash M_i : \alpha_\mathsf{H} \rightarrow \mathtt{bool}$ $(i = 1, 2)$. They have normal forms by Strong Normalization. Since there is no "key", that is, term of α_H under this variable context, we cannot apply M_i to any terms of α_H, so $k : \alpha_\mathsf{L} \vdash M_1 \approx M_2 : \alpha_\mathsf{H} \rightarrow \mathtt{bool}$ by (LL-Term) and (LL-Fun). This example almost corresponds to Example 2.15. In fact, we will translate $[\mathtt{bool}]_\mathsf{H}$ and the observer level H, respectively, to $\alpha_\mathsf{H} \rightarrow \mathtt{bool}$ and $k : \alpha_\mathsf{H}$, in Section 4.

We write δ for a simultaneous substitution of $\lambda^\rightarrow$ terms for variables and $\Delta' \vdash \delta_1 \approx \delta_2 : \Delta$ if $dom(\delta_1) = dom(\delta_2) = dom(\Delta)$ and for any $x \in dom(\delta_1)$, $\Delta' \vdash \delta_1(x) \approx \delta_2(x) : \Delta(x)$. Then, the basic lemma is as follows:

Lemma 3.8 (Basic Lemma). *If $\Delta \vdash M : A$ and $\Delta' \vdash \delta_1 \approx \delta_2 : \Delta$, then $\Delta' \vdash \delta_1(M) \approx \delta_2(M) : A$.*

For later use, we will prove a little generalized lemma as below, from which the basic lemma above follows by reflexivity of $\equiv$ (Lemma 3.4).

Lemma 3.9. *If $\Delta \vdash M_1 \equiv M_2 : A$ and $\Delta' \vdash \delta_1 \approx \delta_2 : \Delta$, then $\Delta' \vdash \delta_1(M_1) \approx \delta_2(M_2) : A$.*

Proof. By induction on the derivation of $\Delta \vdash M_1 \equiv M_2 : A$. We show only the main cases. Below, we write $\delta_1' \uplus \delta_2'$ for the union of two disjoint substitutions δ_1' and δ_2' such that $dom(\delta_1') \cap dom(\delta_2') = \emptyset$: $dom(\delta_1' \uplus \delta_2') = dom(\delta_1') \cup dom(\delta_2')$ and $(\delta_1' \uplus \delta_2')(x) = \delta_i'(x)$ if $x \in dom(\delta_i')$.

Case (the last rule of the derivation is (A-KEY)). Then, the last step of the derivation has a form

$$\frac{\Delta \vdash M_1 : \alpha_\ell \quad \Delta \vdash M_2 : \alpha_\ell}{\Delta \vdash M_1 \equiv M_2 : \alpha_\ell}$$

and $A = \alpha_\ell$. By Substitution Property, Strong Normalization and Subject Reduction, there exists V_i such that $\delta_i(M_i) \rightarrow^* V_i$ and $\Delta' \vdash V_i : \alpha_\ell$ $(i = 1, 2)$. So, since $\Delta' \vdash V_1 \sim V_2 : \alpha_\ell$ by (LL-KT), we get $\Delta' \vdash \delta_1(M_1) \approx \delta_2(M_2) : \alpha_\ell$ by (LL-TERM).

Case (the last rule of the derivation is (A-ABS)). Then, the last step of the derivation has a form

$$\frac{\Delta, \ x : A_1 \vdash M_1' \equiv M_2' : A_2}{\Delta \vdash \lambda x{:}A_1.\,M_1' \equiv \lambda x{:}A_1.\,M_2' : A_1 \rightarrow A_2.}$$

and $M_i = \lambda x{:}A_1.\,M_i'$ $(i = 1, 2)$ and $A = A_1 \rightarrow A_2$. By Strong Normalization, there exist V_i such that $\delta_i(M_i) \longrightarrow^* V_i$ $(i = 1, 2)$. Take arbitrary M_i'' $(i = 1, 2)$ such that $\Delta' \vdash M_1'' \approx M_2'' : A_1$, then $\Delta' \vdash \delta_1 \uplus [M_1''/x] \approx \delta_2 \uplus [M_2''/x] : \Delta \cup \{x : A_1\}$. By the induction hypothesis, $\Delta' \vdash (\delta_1 \uplus [M_1''/x])(M_1') \approx (\delta_2 \uplus [M_2''/x])(M_2') : A_2$. Since $V_i\,M_i''$ have the same normal forms as $(\delta_i \uplus [M_i''/x])(M_i')$ for $i = 1, 2$, we have $\Delta' \vdash V_1\,M_1'' \approx V_2\,M_2'' : A_2$, and hence $\Delta' \vdash V_1 \sim V_2 : A_1 \rightarrow A_2$, so $\Delta' \vdash \delta_1(M_1) \approx \delta_2(M_2) : A_1 \rightarrow A_2$.

Case (the last rule of the derivation is (A-APP)). Then, the last step of the derivation has a form

$$\frac{\Delta \vdash M_1' \equiv M_2' : A_1 \rightarrow A_2 \quad \Delta \vdash M_1'' \equiv M_2'' : A_1}{\Delta \vdash M_1'\,M_1'' \equiv M_2'\,M_2'' : A_2}$$

By the induction hypotheses, $\Delta' \vdash \delta_1(M_1') \approx \delta_2(M_2') : A_1 \rightarrow A_2$ and $\Delta' \vdash \delta_1(M_1'') \approx \delta_2(M_2'') : A_1$. By definition, there exist V_i such that $\delta_i(M_i') \longrightarrow^* V_i$ $(i = 1, 2)$ and $\Delta' \vdash V_1 \sim V_2 : A_1 \rightarrow A_2$, and hence $\Delta' \vdash V_1\,\delta_1(M_1'') \approx V_2\,\delta_2(M_2'') : A_2$. Since $\delta_i(M_i'\,M_i'')$ have the same normal forms as $V_i\,\delta_i(M_i'')$ for $i = 1, 2$, we have $\Delta' \vdash \delta_1(M_1'\,M_1'') \approx \delta_2(M_2'\,M_2'') : A_2$. $\square$

Remark 3.10. Although the above logical relations for $\lambda^\rightarrow$ are not reflexive in general (for example $x : A + A \nvdash x \approx x : A + A$), we have $\Delta \vdash M \approx M : A$ if all the types in Δ are of forms $A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n \rightarrow \alpha_\ell$. This is derived from Lemma 3.8 and the fact that $\Delta \vdash x \approx x : \Delta(x)$ if $\Delta(x) = A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_n \rightarrow \alpha_\ell$, which can be proved by induction on n.

4. TRANSLATION

In this section, we define a formal translation from $\lambda^{[]}$ to $\lambda^\rightarrow$ and its inverse. Both translations are shown to preserve typing.

4.1. **From $\lambda^{[]}$ to $\lambda^{\rightarrow}$.** One of the main ideas of the translation, which closely follows Tse and Zdancewic's translation from DCC to System F [22, 23], is to translate sealing of type $[t]_\ell$ to a function from the base type α_ℓ, which corresponds to ℓ. The sealed value can be extracted by passing a term of α_ℓ as an argument. Intuitively, the term of α_ℓ serves as a "key" for unsealing.

Definition 4.1 (Translation of Types and Contexts). $(\cdot)^\dagger$ is a function from $\lambda^{[]}$ types to $\lambda^{\rightarrow}$ types, defined by:

$$unit^\dagger = unit \qquad (t_1 \text{ op } t_2)^\dagger = t_1^\dagger \text{ op } t_2^\dagger \qquad ([t]_\ell)^\dagger = \alpha_\ell \to t^\dagger$$

where **op** stands for $\rightarrow, \times,$ or $+$. $(\cdot)^\dagger$ is extended pointwise to contexts by: $\Gamma^\dagger = \{x : t^\dagger \mid x : t \in \Gamma\}$.

Before describing the details of the translation, we give an example for readers to grasp its intuition.

Example 4.2. We translate the $\lambda^{[]}$ judgment $x : [\texttt{bool}]_\texttt{L} ; \texttt{H} \vdash x^\texttt{L} : \texttt{bool}$ to:

$$x : \alpha_\texttt{L} \to \texttt{bool}, c_\texttt{LL} : \alpha_\texttt{L} \to \alpha_\texttt{L}, c_\texttt{HH} : \alpha_\texttt{H} \to \alpha_\texttt{H}, c_\texttt{HL} : \alpha_\texttt{H} \to \alpha_\texttt{L}, k_\texttt{H} : \alpha_\texttt{H} \vdash x\,(c_\texttt{HL}\,k_\texttt{H}) : \texttt{bool}.$$

The first and last variable declarations are respectively translated results of $x : [\texttt{bool}]_\texttt{L}$ and the observer level $\texttt{H}$. The unsealing $x^\texttt{L}$ is translated into the application of x to $c_\texttt{HL}\,k_\texttt{H}$ which corresponds to a key for the unsealing, and where $c_\texttt{HL}$ coerces the key $k_\texttt{H}$ for the observer level $\texttt{H}$ to that for $\texttt{L}$. This coercion is declared at the second last variable declaration. The other variables $c_\texttt{LL}$ and $c_\texttt{HH}$ are trivial coercions.

Let c be an injective partial map from pairs of levels to variables such that $c_{\ell_2\,\ell_1}$ is defined if and only if $\ell_1 \sqsubseteq \ell_2$. We take a finite mapping $C_\sqsubseteq = \{c_{\ell_2\,\ell_1} : \alpha_{\ell_2} \to \alpha_{\ell_1} \mid \ell_1 \sqsubseteq \ell_2\}$ from variables to types, which corresponds to the variable declarations

$$c_\texttt{LL} : \alpha_\texttt{L} \to \alpha_\texttt{L}, c_\texttt{HH} : \alpha_\texttt{H} \to \alpha_\texttt{H}, c_\texttt{HL} : \alpha_\texttt{H} \to \alpha_\texttt{L}$$

in Example 4.2. Each variable $c_{\ell_2\,\ell_1}$ represents a function to coerce a key for a higher level to that for a lower. As like above, $C_\sqsubseteq$ will be included in a variable context for typing the translated terms. Note that, if we let $\mathcal{L}$ be infinite, the domain of $C_\sqsubseteq$ would be so, too, and hence we would have to extend the type judgments of $\lambda^{\rightarrow}$ to allow an infinite context. Such an extension would be easy since only a finite number of variables can be used in a term.

The translation of $\lambda^{[]}$ to $\lambda^{\rightarrow}$ is represented by $\Gamma; \sigma \vdash e : t \searrow M$, read "$\lambda^{[]}$ term e of type t is translated to M under Γ and σ," where σ is an injective finite map from data levels to variables. In the example above, σ is $\{\texttt{H} \mapsto k_\texttt{H}\}$. This mapping σ, whose domain represents the observer level at which the $\lambda^{[]}$ term is typed, records correspondence between the data levels included in the observer level and variables that are used as keys. When typing the translated term in $\lambda^{\rightarrow}$, those variables are declared in the variable context (e.g., $k_\texttt{H} : \alpha_\texttt{H}$ in Example 4.2), and hence, from usual conventions of $\lambda^{\rightarrow}$, we assume that the range of σ and the domains of Γ and $C_\sqsubseteq$ are pairwise disjoint and that we can implicitly rename variables in the range of σ, so that choices for key names do not matter.

Definition 4.3 (Translation of Terms). The relation $\Gamma; \sigma \vdash e : t \searrow M$ is defined as the least relation closed under the following rules:

$$\Gamma; \sigma \vdash x : t \searrow x \tag{TR-VAR}$$

$$\Gamma; \sigma \vdash () : unit \searrow () \tag{TR-UNIT}$$

$$\frac{\Gamma, x : t_1; \sigma \vdash e : t_2 \searrow M}{\Gamma; \sigma \vdash \lambda x{:}t_1.\, e : t_1 \to t_2 \searrow \lambda x{:}t_1^\dagger.\, M} \tag{Tr-Abs}$$

$$\frac{\Gamma; \sigma \vdash e : t_1 \to t_2 \searrow M \qquad \Gamma; \sigma \vdash e' : t_1 \searrow M'}{\Gamma; \sigma \vdash e\, e' : t_2 \searrow M\, M'} \tag{Tr-App}$$

$$\frac{\Gamma; \sigma \vdash e_1 : t_1 \searrow M_1 \qquad \Gamma; \sigma \vdash e_2 : t_2 \searrow M_2}{\Gamma; \sigma \vdash \langle e_1, e_2 \rangle : t_1 \times t_2 \searrow \langle M_1, M_2 \rangle} \tag{Tr-Pair}$$

$$\frac{\Gamma; \sigma \vdash e : t_1 \times t_2 \searrow M \qquad i \in \{1, 2\}}{\Gamma; \sigma \vdash \pi_i(e) : t_i \searrow \pi_i(M)} \tag{Tr-Proj}$$

$$\frac{\Gamma; \sigma \vdash e : t_i \searrow M \qquad i \in \{1, 2\}}{\Gamma; \sigma \vdash \iota_i(e) : t_1 + t_2 \searrow \iota_i(M)} \tag{Tr-Inj}$$

$$\frac{\Gamma; \sigma \vdash e : t_1 + t_2 \searrow M \qquad \Gamma, x_1 : t_1; \sigma \vdash e_1 : t \searrow M_1 \qquad \Gamma, x_2 : t_2; \sigma \vdash e_2 : t \searrow M_2}{\Gamma; \sigma \vdash (\textbf{case}\, e\, \textbf{of}\, \iota_1(x_1).e_1 \mid \iota_2(x_2).e_2) : t \searrow (\textbf{case}\, M\, \textbf{of}\, \iota_1(x_1).M_1 \mid \iota_2(x_2).M_2)}$$
$$\tag{Tr-Case}$$

$$\frac{\Gamma; \sigma\{\ell \mapsto k\} \vdash e : t \searrow M \qquad k\ \text{fresh}}{\Gamma; \sigma \vdash [e]_\ell : [t]_\ell \searrow \lambda k{:}\alpha_\ell.\, M} \tag{Tr-Seal}$$

$$\frac{\Gamma; \sigma \vdash e : [t]_\ell \searrow M \qquad \ell' \in dom(\sigma) \qquad \ell \sqsubseteq \ell'}{\Gamma; \sigma \vdash e^\ell : t \searrow M\, (c_{\ell'\, \ell}\, \sigma(\ell'))} \tag{Tr-Unseal}$$

Here, we write $\sigma\{\ell \mapsto k\}$ for a mapping from $dom(\sigma) \cup \{\ell\}$ to variables defined by: $\sigma\{\ell \mapsto k\}(\ell) = k$; and $\sigma\{\ell \mapsto k\}(\ell') = \sigma(\ell')$ if $\ell \neq \ell'$. Note that ℓ may occur in the domain of σ.

The translation of terms is easily derived from the translation rules for types. In the last rule (Tr-Unseal), a key for opening the sealing is obtained from σ and a coercion—if e^ℓ is well typed at the observer level represented by $dom(\sigma)$, then ℓ should be lower than $dom(\sigma)$ and hence a coercion function should exist in $C_\sqsubseteq$ to provide a key of ℓ.

Example 4.4. Let L and H_1 and H_2 be data levels and suppose that L is strictly lower than both H_1 and H_2. We can translate $x : [\texttt{bool}]_\text{L}; \text{H}_1, \text{H}_2 \vdash [x^\text{L}]_{\text{H}_1} : [\texttt{bool}]_{\text{H}_1}$ as follows:

$$x : [\texttt{bool}]_\text{L}; \{\text{H}_1 \mapsto k_1, \text{H}_2 \mapsto k_2\} \vdash [x^\text{L}]_{\text{H}_1} : [\texttt{bool}]_{\text{H}_1} \searrow \lambda k_1'{:}\alpha_{\text{H}_1} \to \texttt{bool}.\, x\, K$$

where K is $c_{\text{H}_2\, \text{L}}\, k_2$ or $c_{\text{H}_1\, \text{L}}\, k_1'$, but not $c_{\text{H}_1\, \text{L}}\, k_1$ because of the side condition of (Tr-Seal). The resulting $\lambda^\to$ terms have type $\alpha_{\text{H}_1} \to \texttt{bool}(= [\texttt{bool}]_{\text{H}_1}^\dagger)$ under context

$$\Delta_0 \overset{\text{def}}{=} x : \alpha_\text{L} \to \texttt{bool}, C_\sqsubseteq, k_1 : \text{H}_1, k_2 : \text{H}_2\ .$$

Well typed $\lambda^{[]}$ terms can be translated to well typed $\lambda^\to$ terms as in the theorem below. Here, we write $\sigma^\dagger$ for the context defined by: $\{\sigma(\ell) : \alpha_\ell \mid \ell \in dom(\sigma)\}$.

Theorem 4.5 (Translation Preserves Typing). *If $\Gamma; \pi \vdash e : t$ and $dom(\sigma) = \pi$, then there exists a $\lambda^\to$ term M such that $\Gamma; \sigma \vdash e : t \searrow M$, and that $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M : t^\dagger$.*

Proof. By induction on the derivation of $\Gamma; \pi \vdash e : t$. We show only the main cases:

Case (the last rule of the derivation is (ST-Seal)). Then, $e = [e_0]_\ell$ and $t = [t_0]_\ell$ for some e_0 and t_0. Take a fresh variable k such that $ran(\sigma\{\ell \mapsto k\}) \cap dom(\Gamma) = \emptyset$. By the induction hypothesis, there exists M_0 such that $\Gamma; \sigma\{\ell \mapsto k\} \vdash e_0 : t_0 \searrow M_0$ and $\Gamma^\dagger, C_\sqsubseteq, (\sigma\{\ell \mapsto k\})^\dagger \vdash M_0 : t_0^\dagger$. Note that $(\sigma\{\ell \mapsto k\})^\dagger = \sigma^\dagger\backslash\{\sigma(\ell) : \alpha_\ell\} \cup \{k : \alpha_\ell\}$. Hence, $\Gamma; \sigma \vdash [e_0]_\ell : [t_0]_\ell \searrow \lambda k{:}\alpha_\ell. M_0$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash \lambda k{:}\alpha_\ell. M_0 : t_0^\dagger$ by (LT-Abs) and weakening.

Case (the last rule of the derivation is (ST-Unseal)). Then, $e = e_0^\ell$ for some e_0. By the induction hypothesis, there exists M_0 such that $\Gamma; \sigma \vdash e_0 : [t]_\ell \searrow M_0$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_0 : \alpha_\ell \to t^\dagger$. Note that $\ell \sqsubseteq \ell' \in \pi = dom(\sigma)$, so $\Gamma; \sigma \vdash e_0^\ell : t \searrow M_0 (c_{\ell' \ell} \sigma(\ell'))$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M (c_{\ell' \ell} \sigma(\ell')) : t^\dagger$.

The other cases are similar. $\qquad\square$

Note that, as we have seen in Example 4.4, the translation result might not be unique since there might be many keys to be coerced to one for some observer level in applying (Tr-Unseal). In fact, if we can translate an unsealing term with some key included in σ, where another higher key exists, then, another translation is also possible by using the latter key instead of the former one, which may be removed from σ. This fact is generalized as follows.

Lemma 4.6. *Assume that* $\Gamma; \sigma\{\ell_1 \mapsto k_1\} \vdash e : t \searrow M$ *and that* $\ell_1 \sqsubseteq \ell_2 \in dom(\sigma)$. *Then, there exists M' such that $\Gamma; \sigma \vdash e : t \searrow M'$ and, if $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_1 : \alpha_{\ell_1}$, then $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash [M_1/k_1]M \equiv M' : t^\dagger$. The sizes of the derivations of the translations are the same.*

Proof. By induction on the size of the derivation of $\Gamma; \sigma\{\ell_1 \mapsto k_1\} \vdash e : t \searrow M$. Note that every occurrence of k_1 in M appears as $c_{\ell_1 \ell} k_1$ for some ℓ, since k_1 is always introduced by (Tr-Unseal). Because σ has the higher key of α_{ℓ_2} than k_1, we can replace all the $c_{\ell_1 \ell} k_1$ and remove all the occurrences of k_1. The last equivalence follows from (A-Key). $\qquad\square$

4.2. **From $\lambda^\to$ to $\lambda^{[]}$.** We define the inverse translation, represented by $\Gamma; \sigma \vdash M \nearrow e : t$. It is read "$\lambda^\to$ term M of type $t^\dagger$ under $\Gamma^\dagger$ and $C_\sqsubseteq$ and $\sigma^\dagger$ is translated back to a $\lambda^{[]}$ term e."

Definition 4.7 (Inverse Translation). The relation $\Gamma; \sigma \vdash M \nearrow e : t$ is defined as the least relation closed by the following rules:

$$\Gamma; \sigma \vdash x \nearrow x : t \tag{ITr-Var}$$

$$\Gamma; \sigma \vdash () \nearrow () : unit \tag{ITr-Unit}$$

$$\frac{\Gamma, x : t_1; \sigma \vdash M \nearrow e : t_2}{\Gamma; \sigma \vdash \lambda x{:}t_1^\dagger. M \nearrow \lambda x{:}t_1. e : t_1 \to t_2} \tag{ITr-Abs}$$

$$\frac{\Gamma; \sigma \vdash M \nearrow e : t_1 \to t_2 \qquad \Gamma; \sigma \vdash M' \nearrow e' : t_1}{\Gamma; \sigma \vdash M M' \nearrow e\, e' : t_2} \tag{ITr-App}$$

$$\frac{\Gamma; \sigma \vdash M_1 \nearrow e_1 : t_1 \qquad \Gamma; \sigma \vdash M_2 \nearrow e_2 : t_2}{\Gamma; \sigma \vdash \langle M_1, M_2 \rangle \nearrow \langle e_1, e_2 \rangle : t_1 \times t_2} \tag{ITr-Pair}$$

$$\frac{\Gamma; \sigma \vdash M \nearrow e : t_1 \times t_2 \qquad i \in \{1, 2\}}{\Gamma; \sigma \vdash \pi_i(M) \nearrow \pi_i(e) : t_i} \qquad \text{(ITR-PROJ)}$$

$$\frac{\Gamma; \sigma \vdash M \nearrow e : t_i \qquad i \in \{1, 2\}}{\Gamma; \sigma \vdash \iota_i(M) \nearrow \iota_i(e) : t_1 + t_2} \qquad \text{(ITR-INJ)}$$

$$\frac{\Gamma; \sigma \vdash M \nearrow e : t_1 + t_2 \qquad \Gamma, x_1 : t_1; \sigma \vdash M_1 \nearrow e_1 : t \qquad \Gamma, x_2 : t_2; \sigma \vdash M_2 \nearrow e_2 : t}{\Gamma; \sigma \vdash (\textbf{case } M \textbf{ of } \iota_1(x_1).M_1 \mid \iota_2(x_2).M_2) \nearrow (\textbf{case } e \textbf{ of } \iota_1(x_1).e_1 \mid \iota_2(x_2).e_2) : t} \qquad \text{(ITR-CASE)}$$

$$\frac{\ell \notin \mathit{dom}(\sigma) \qquad \Gamma; \sigma\{\ell \mapsto k\} \vdash M \nearrow e : t}{\Gamma; \sigma \vdash \lambda k{:}\alpha_\ell.\, M \nearrow [e]_\ell : [t]_\ell} \qquad \text{(ITR-SEAL1)}$$

$$\frac{\ell \in \mathit{dom}(\sigma) \qquad \Gamma; \sigma\{\ell \mapsto k\} \vdash [k/\sigma(\ell)]M \nearrow e : t}{\Gamma; \sigma \vdash \lambda k{:}\alpha_\ell.\, M \nearrow [e]_\ell : [t]_\ell} \qquad \text{(ITR-SEAL2)}$$

$$\frac{\Gamma; \sigma \vdash M \nearrow e : [t]_\ell \qquad \Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M' : \alpha_\ell}{\Gamma; \sigma \vdash M\, M' \nearrow e^\ell : t} \qquad \text{(ITR-UNSEAL)}$$

In the rule (ITR-SEAL2), since we equate keys for the same data level by (A-KEY) and (LL-KT), we can replace the key $\sigma(\ell)$ by another k. Note that, even if $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M : t^\dagger$, the inverse translation of M is *not* always possible. However, we can give a sufficient condition for the inverse translation to exist and show that the inverse translation also preserves typing:

Theorem 4.8 (Inverse Translation Preserves Typing)**.** *If all the subderivations of* $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M : t^\dagger$ *satisfy SUbformula Property, then there exists a* $\lambda^{[]}$ *term e such that* $\Gamma; \mathit{dom}(\sigma) \vdash e : t$ *and* $\Gamma; \sigma \vdash M \nearrow e : t$.

Proof. By induction on the size of the derivation of $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M : t^\dagger$. We show only the main cases:

Case (the last rule of the derivation is (LT-ABS))**.** Then, the last step of the derivation has a form

$$\frac{\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger, x : A_1 \vdash M_0 : A_2}{\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash \lambda x{:}A_1.\, M_0 : A_1 \to A_2,}$$

and $t^\dagger = A_1 \to A_2$ and $M = \lambda x{:}A_1.\, M_0$. We have three subcases:

Subcase $(t = t_1 \to t_2)$**.** Then, $t_i^\dagger = A_i (i = 1, 2)$ and $\Gamma^\dagger, x : t_1^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_0 : t_2^\dagger$, all the subderivations of which also satisfy Subformula Property. So, by the induction hypothesis, there exists e_0 such that $\Gamma, x : t_1; \mathit{dom}(\sigma) \vdash e_0 : t_2$ and $\Gamma, x : t_1; \sigma \vdash M_0 \nearrow e_0 : t_2$. Hence, $\Gamma; \mathit{dom}(\sigma) \vdash \lambda x{:}t_1.e_0 : t_1 \to t_2$ and $\Gamma; \sigma \vdash \lambda x{:}A_1.\, M_0 \nearrow \lambda x{:}t_1.\, e_0 : t_1 \to t_2$.

Subcase $(t = [t_0]_\ell$ and $\ell \notin \mathit{dom}(\sigma))$**.** Then, $A_1 = \alpha_\ell$ and $A_2 = t_0^\dagger$ and $(\sigma\{\ell \mapsto x\})^\dagger = \sigma^\dagger \cup \{x : \alpha_\ell\}$ and $\Gamma^\dagger, C_\sqsubseteq, (\sigma\{\ell \mapsto x\})^\dagger \vdash M_0 : t_0^\dagger$, all the subderivations of which also satisfy Subformula Property. So, by the induction hypothesis, there exists e_0 such that $\Gamma; \mathit{dom}(\sigma\{\ell \mapsto x\}) \vdash e_0 : t_0$ and $\Gamma; \sigma\{\ell \mapsto x\} \vdash M_0 \nearrow e_0 : t_0$. Since $\ell \notin \mathit{dom}(\sigma)$ and $\mathit{dom}(\sigma\{\ell \mapsto x\}) = \mathit{dom}(\sigma) \cup \{\ell\}$, it follows that $\Gamma; \mathit{dom}(\sigma) \vdash [e_0]_\ell : [t_0]_\ell$ by (ST-SEAL) and $\Gamma; \sigma \vdash \lambda x{:}\alpha_\ell.\, M_0 \nearrow [e_0]_\ell : [t_0]_\ell$ by (ITR-SEAL1).

Subcase $(t = [t_0]_\ell$ **and** $\ell \in dom(\sigma))$**.** Then, $A_1 = \alpha_\ell$ and $A_2 = t_0^\dagger$ and $(\sigma\{\ell \mapsto x\})^\dagger = \sigma^\dagger \backslash \{\sigma(\ell) : \alpha_\ell\} \cup \{x : \alpha_\ell\}$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger, x : \alpha_\ell \vdash M_0 : t_0^\dagger$. By Substitution Property for $\lambda^\rightarrow$, $\Gamma^\dagger, C_\sqsubseteq, (\sigma\{\ell \mapsto x\})^\dagger \vdash [x/\sigma(\ell)]M_0 : t_0^\dagger$ without changing the size of the derivation, all the subderivations of which also satisfy Subformula Property. So, by the induction hypothesis, there exists a e_0 such that $\Gamma; dom(\sigma\{\ell \mapsto x\}) \vdash e_0 : t_0$ and $\Gamma; \sigma\{\ell \mapsto x\} \vdash [x/\sigma(\ell)]M_0 \nearrow e_0 : t_0$. Since $dom(\sigma\{\ell \mapsto x\}) = dom(\sigma) \cup \{\ell\}$ and $\ell \in dom(\sigma)$, it follows that $\Gamma; dom(\sigma) \vdash [e_0]_\ell : [t_0]_\ell$ by (ST-SEAL) and $\Gamma; \sigma \vdash \lambda x : \alpha_\ell. M_0 \nearrow [e_0]_\ell : [t_0]_\ell$ by (ITR-SEAL2).

Case (the last rule of the derivation is (LT-APP)). Then, the last step of the derivation has a form

$$\frac{\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_1 : A_1 \rightarrow A_2 \quad \Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_2 : A_1}{\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_1 M_2 : A_2}$$

and $t^\dagger = A_2$ and $M = M_1 M_2$. By Subformula Property, A_1 and $A_1 \rightarrow A_2$ appear in $\Gamma^\dagger \cup C_\sqsubseteq \cup \sigma^\dagger \cup t^\dagger$, hence, we have two cases about A_1: $A_1 = \alpha_\ell$ or $A_1 = t_0^\dagger$ for some t_0.

Subcase $(A_1 = \alpha_\ell)$**.** Then, $A_1 \rightarrow A_2 = ([t]_\ell)^\dagger$, by the induction hypothesis, there exists e such that $\Gamma; dom(\sigma) \vdash e : [t]_\ell$ and $\Gamma; \sigma \vdash M_1 \nearrow e : [t]_\ell$. Note that $\ell \sqsubseteq dom(\sigma)$ since $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_2 : \alpha_\ell$. So, it follows that $\Gamma; dom(\sigma) \vdash e^\ell : t$ and $\Gamma; \sigma \vdash M_1 M_2 \nearrow e^\ell : t$ by (ST-UNSEAL) and (ITR-UNSEAL).

Subcase $(A_1 = t_0^\dagger)$**.** Then, $A_1 \rightarrow A_2 = (t_0 \rightarrow t_1)^\dagger$. By the induction hypotheses, we can easily show the conclusion.

For the cases where the last rule of the derivation is an elimination of a product or sum type, the proof is similar to the case of application. The rest of the proof is easy. □

Remark 4.9. In the above theorem, Subformula Property gives a sufficient condition to exclude "junk" terms such as $(\lambda x : \alpha_\ell \rightarrow \alpha_\ell. ())(\lambda k : \alpha_\ell. k)$. Since $\lambda k : \alpha_\ell. k$ has type $\alpha_\ell \rightarrow \alpha_\ell$, no rules of inverse translation can be applied and the inverse translation will fail. Its derivation, however, does not satisfy Subformula Property, so this is not a counterexample for the theorem above. (In fact, its normal form can be translated back to a $\lambda^{[]}$ term.)

Example 4.10. We use the same settings as Example 4.4.

$$x : [\texttt{bool}]_\texttt{L}; \{\texttt{H}_1 \mapsto k_1, \texttt{H}_2 \mapsto k_2\} \vdash \lambda k_1' : \alpha_{\texttt{H}_1} \rightarrow \texttt{bool}. \, x \, K \nearrow [x^\texttt{L}]_{\texttt{H}_1} : [\texttt{bool}]_{\texttt{H}_1}$$

where K can be any term of type $\alpha_\texttt{L}$ under context Δ_0, $k_1' : \alpha_{\texttt{H}_1} \rightarrow \texttt{bool}$, e.g, $c_{\texttt{H}_2 \, \texttt{L}} \, k_2$ or $c_{\texttt{H}_1 \, \texttt{L}} \, k_1'$ or $c_{\texttt{H}_1 \, \texttt{L}} \, k_1$.

5. Proof of Noninterference via Preservation of Logical Relations

In this section, we give an indirect proof of the noninterference theorem, which is obtained as an easy corollary of the theorem that the translation is sound and complete, that is, the logical relation for $\lambda^{[]}$ is preserved and reflected by the translation to $\lambda^\rightarrow$. The properties we would expect are

If $\cdot; \sigma \vdash e_i : t \searrow M_i$ for $i = 1, 2$ and $e_1 \approx_{dom(\sigma)} e_2 : t$, then $C_\sqsubseteq, \sigma^\dagger \vdash M_1 \approx M_2 : t^\dagger$,

and its converse

$\quad$ If $\cdot; \sigma \vdash e_i : t \searrow M_i$ for $i = 1, 2$ and $C_{\sqsubseteq}, \sigma^{\dagger} \vdash M_1 \approx M_2 : t^{\dagger}$, then
$\quad$ $e_1 \approx_{dom(\sigma)} e_2 : t$.

It is not very easy, however, to prove them directly because logical relations are defined by induction on types whereas the translations are not. Thus, following Tse and Zdancewic [21, 22, 23], we introduce another logical relation (called *logical correspondence*) $e \approx\!\!\!\approx_{\sigma} M : t$ over terms of $\lambda^{[]}$ and $\lambda^{\rightarrow}$, then prove that it includes (the graphs of) the translations of both directions (Theorems 5.4 and 5.6). Then, after showing that the logical correspondence is full (Corollary 5.7), we finally prove preservation of logical relations by logical correspondence and reduce the noninterference theorem to Basic Lemma (Lemma 3.8).

5.1. Logical Correspondence and Its Fullness.

Definition 5.1 (Logical Correspondence)**.** The relations $e \approx\!\!\!\approx_{\sigma} M : t$ and $v \rightsquigarrow_{\sigma} V : t$, where we assume that $\cdot; dom(\sigma) \vdash e : t$ and $\cdot; dom(\sigma) \vdash v : t$ and $C_{\sqsubseteq}, \sigma^{\dagger} \vdash M : t^{\dagger}$ and $C_{\sqsubseteq}, \sigma^{\dagger} \vdash V : t^{\dagger}$, are defined as the least relation closed under the following rules:

$$() \rightsquigarrow_{\sigma} () : unit \qquad\qquad\qquad \text{(C-Unit)}$$

$$\frac{\forall (e \approx\!\!\!\approx_{\sigma} M : t_1). \, v \, e \approx\!\!\!\approx_{\sigma} V \, M : t_2}{v \rightsquigarrow_{\sigma} V : t_1 \rightarrow t_2} \qquad\qquad \text{(C-Fun)}$$

$$\frac{v_1 \rightsquigarrow_{\sigma} V_1 : t_1 \qquad v_2 \rightsquigarrow_{\sigma} V_2 : t_2}{\langle v_1, \, v_2 \rangle \rightsquigarrow_{\sigma} \langle V_1, \, V_2 \rangle : t_1 \times t_2} \qquad\qquad \text{(C-Pair)}$$

$$\frac{v \rightsquigarrow_{\sigma} V : t_i \qquad i \in \{1, \, 2\}}{\iota_i(v) \rightsquigarrow_{\sigma} \iota_i(V) : t_1 + t_2} \qquad\qquad \text{(C-Inj)}$$

$$\frac{\forall (C_{\sqsubseteq}, \, \sigma^{\dagger} \vdash M : \alpha_{\ell}). \, v \approx\!\!\!\approx_{\sigma} V \, M : t}{[v]_{\ell} \rightsquigarrow_{\sigma} V : [t]_{\ell}} \qquad\qquad \text{(C-Seal)}$$

$$\frac{e \longrightarrow^{*} v \qquad M \longrightarrow^{*} V \qquad v \rightsquigarrow_{\sigma} V : t}{e \approx\!\!\!\approx_{\sigma} M : t} \qquad\qquad \text{(C-Term)}$$

$\quad$ Intuitively, $e \approx\!\!\!\approx_{\sigma} M : t$ means that e and M exhibit the same behavior from the viewpoint of an observer at $dom(\sigma)$. The rule (C-Seal) for $[t]_{\ell}$ expresses the fact that the existence of well typed M of α_{ℓ} under $C_{\sqsubseteq}$ and $\sigma^{\dagger}$ is equivalent to the fact that the level ℓ is lower than $dom(\sigma)$. In other words, if ℓ is not lower than $dom(\sigma)$, the premise is vacuously true, representing that the observer cannot distinguish anything.

Example 5.2. Take $\lambda^{[]}$ term e and $\lambda^{\rightarrow}$ term M such that $\cdot; \mathtt{L} \vdash e : [\mathtt{bool}]_{\mathtt{H}}$ and $C_{\sqsubseteq}, k : \alpha_{\mathtt{L}} \vdash M : \alpha_{\mathtt{H}} \rightarrow \mathtt{bool}$. By (C-Term) and (C-Seal), $e \approx\!\!\!\approx_{\{\mathtt{L} \mapsto k\}} M : [\mathtt{bool}]_{\mathtt{H}}$ because there is no term of type $\alpha_{\mathtt{H}}$ under $C_{\sqsubseteq}, k : \alpha_{\mathtt{L}}$. Compare this example with Examples 2.15 and 3.7.

$\quad$ Theorem 5.3 below shows that the logical correspondences are closed under the composition with the logical relations in $\lambda^{\rightarrow}$.

Theorem 5.3. If $e \approx\!\!\!\approx_{\sigma} M_1 : t$ and $C_{\sqsubseteq}, \sigma^{\dagger} \vdash M_1 \approx M_2 : t^{\dagger}$, then $e \approx\!\!\!\approx_{\sigma} M_2 : t$.

Proof. By induction on the structure of t. We show only the main cases:

Case ($t = t_1 \to t_2$). By definition, there exist v and V_i such that $e \longrightarrow^* v$ and $M_i \longrightarrow^* V_i$ ($i = 1, 2$) and $v \rightsquigarrow_\sigma V_1 : t_1 \to t_2$ and $C_{\sqsubseteq}, \sigma^\dagger \vdash V_1 \sim V_2 : t_1^\dagger \to t_2^\dagger$. Take arbitrary e_0 and M_0 such that $e_0 \approx\!\!\!\approx_\sigma M_0 : t_1$. By definition, $v\, e_0 \approx\!\!\!\approx_\sigma V_1\, M_0 : t_2$. Also, by Lemma 3.8 (with Remark 3.10), $C_{\sqsubseteq}, \sigma^\dagger \vdash M_0 \approx M_0 : t_1^\dagger$, so, by definition, $C_{\sqsubseteq}, \sigma^\dagger \vdash V_1\, M_0 \approx V_2\, M_0 : t_2^\dagger$. Applying the induction hypothesis for t_2, we have $v\, e_0 \approx\!\!\!\approx_\sigma V_2\, M_0 : t_2^\dagger$ and hence $v \rightsquigarrow_\sigma V_2 : t_1 \to t_2$, so $e \approx\!\!\!\approx_\sigma M_2 : t_1 \to t_2$.

Case ($t = [t_1]_\ell$). By definition, there exist v and V_i such that $e \longrightarrow^* [v]_\ell$ and $M_i \longrightarrow^* V_i$ ($i = 1, 2$) and $[v]_\ell \rightsquigarrow_\sigma V_1 : [t_1]_\ell$ and $C_{\sqsubseteq}, \sigma^\dagger \vdash V_1 \approx V_2 : \alpha_\ell \to t_1^\dagger$. Take arbitrary M_0 such that $C_{\sqsubseteq}, \sigma^\dagger \vdash M_0 : \alpha_\ell$. By definition, $v \approx\!\!\!\approx_\sigma V_1\, M_0 : t_1$ and $C_{\sqsubseteq}, \sigma^\dagger \vdash M_0 \approx M_0 : \alpha_\ell$, so, $C_{\sqsubseteq}, \sigma^\dagger \vdash V_1\, M_0 \approx V_2\, M_0 : t_1^\dagger$. Applying the induction hypothesis for t_1, we have $v \approx\!\!\!\approx_\sigma V_2\, M_0 : t_1$ and hence $[v]_\ell \rightsquigarrow_\sigma V_2 : [t_1]_\ell$, so, $e \approx\!\!\!\approx_\sigma M_2 : [t_1]_\ell$. $\qquad\square$

The next theorem shows that these logical correspondences include the graphs of the translation to $\lambda^\to$. We write $\gamma \approx\!\!\!\approx_\sigma \delta : \Gamma$ if $dom(\gamma) = dom(\delta) = dom(\Gamma)$ and $\gamma(x) \approx\!\!\!\approx_\sigma \delta(x) : \Gamma(x)$ for any $x \in dom(\Gamma)$.

Theorem 5.4 (Inclusion of Translation). *If $\Gamma; dom(\sigma) \vdash e : t$ and $\Gamma; \sigma \vdash e : t \searrow M$ and $\gamma \approx\!\!\!\approx_\sigma \delta : \Gamma$, then $\gamma(e) \approx\!\!\!\approx_\sigma \delta(M) : t$.*

Proof. By induction on the size of the derivation of $\Gamma; \sigma \vdash e : t \searrow M$. We show only the main cases:

Case (the last translation rule of the derivation is (TR-ABS)). Then, the last step of the derivation has a form

$$\frac{\Gamma, x : t_1; \sigma \vdash e_0 : t_2 \searrow M_0}{\Gamma; \sigma \vdash \lambda x{:}t_1.\, e_0 : t_1 \to t_2 \searrow \lambda x{:}t_1^\dagger.\, M_0.}$$

Take arbitrary e_1 and M_1 such that $e_1 \approx\!\!\!\approx_\sigma M_1 : t_1$, then, $\gamma \uplus [e_1/x] \approx\!\!\!\approx_\sigma \delta \uplus [M_1/x] : \Gamma \cup \{x : t_1\}$. By the induction hypothesis, $(\gamma \uplus [e_1/x])(e_0) \approx\!\!\!\approx_\sigma (\delta \uplus [M_1/x])(M_0) : t_2$. Since $\gamma(\lambda x{:}t_1.\, e_0)\, e_1$ and $\delta(\lambda x{:}t_1^\dagger.\, M_0)\, M_1$ have the same normal forms as $(\gamma \uplus [e_1/x])(e_0)$ and $(\delta \uplus [M_1/x])(M_0)$, respectively, we have $\gamma(\lambda x{:}t_1.\, e_0)\, e_1 \approx\!\!\!\approx_\sigma \delta(\lambda x{:}t_1^\dagger.\, M_0)\, M_1 : t_2$, and hence $\gamma(\lambda x{:}t_1.\, e_0) \approx\!\!\!\approx_\sigma \delta(\lambda x{:}t_1^\dagger.\, M_0) : t_1 \to t_2$.

Case (the last translation rule of the derivation is (TR-APP)). Then, the last step of the derivation has a form

$$\frac{\Gamma; \sigma \vdash e_1 : t_1 \to t_2 \searrow M_1 \quad \Gamma; \sigma \vdash e_2 : t_1 \searrow M_2}{\Gamma; \sigma \vdash e_1\, e_2 : t_2 \searrow M_1\, M_2}.$$

By the induction hypotheses, $\gamma(e_1) \approx\!\!\!\approx_\sigma \delta(M_1) : t_1 \to t_2$ and $\gamma(e_2) \approx\!\!\!\approx_\sigma \delta(M_2) : t_1$. By Strong Normalization, $\gamma(e_1)$ and $\delta(M_1)$ respectively have the unique normal forms v and V such that $v \rightsquigarrow_\sigma V : t_1 \to t_2$. By definition, we have $v\, \gamma(e_2) \approx\!\!\!\approx_\sigma V\, \delta(M_2) : t_2$ and hence $\gamma(e_1\, e_2) \approx\!\!\!\approx_\sigma \delta(M_1\, M_2) : t_2$.

Case (the last translation rule of the derivation is (TR-SEAL)). Then, the last step of the derivation has a form

$$\frac{\Gamma; \sigma\{\ell \mapsto k\} \vdash e_0 : t_0 \searrow M_0 \qquad k \text{ fresh}}{\Gamma; \sigma \vdash [e_0]_\ell : [t_0]_\ell \searrow \lambda k{:}\alpha_\ell.\, M_0}.$$

Then, there exist v and V such that $\gamma(e_0) \longrightarrow^* v$ and $\delta(\lambda k{:}\alpha_\ell.\, M_0) \longrightarrow^* V$. Take arbitrary M_1 such that $C_\sqsubseteq, \sigma^\dagger \vdash M_1 : \alpha_\ell$. Then there exists $\ell' \in dom(\sigma)$ such that $\ell \sqsubseteq \ell'$ and, by Lemma 4.6, there exists M_0' such that $\Gamma; \sigma \vdash e_0 : t_0 \searrow M_0'$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_0' \equiv [M_1/k]M_0 : t_0^\dagger$. So, by the induction hypothesis, $\gamma(e_0) \approx\!\!\approx_\sigma \delta(M_0') : t_0$. Also, by Lemma 3.9, we have $C_\sqsubseteq, \sigma^\dagger \vdash \delta(M_0') \approx \delta([M_1/k]M_0) : t_0^\dagger$. Since $\delta([M_1/k]M_0)$ and $\delta(\lambda k{:}\alpha_\ell.\, M_0)M_1$ have the same normal form, $C_\sqsubseteq, \sigma^\dagger \vdash \delta(M_0') \approx \delta(\lambda k{:}\alpha_\ell.\, M_0)M_1 : t_0^\dagger$, and, applying Theorem 5.3, we get $\gamma(e_0) \approx\!\!\approx_\sigma \delta(\lambda k{:}\alpha_\ell.\, M_0)\ M_1 : t_0$, hence $v \approx\!\!\approx_\sigma V\ M_1 : t_0$, so $[v]_\ell \leadsto_\sigma V : [t_0]_\ell$. Therefore $\gamma([e_0]_\ell) \approx\!\!\approx_\sigma \delta(\lambda k{:}\alpha_\ell.\, M_0) : [t_0]_\ell$.

Case (the last translation rule of the derivation is (TR-UNSEAL)). Assume that the last step of the derivation has a form

$$\frac{\Gamma; \sigma \vdash e_1 : [t_1]_\ell \searrow M_1 \quad \ell' \in dom(\sigma) \quad \ell \sqsubseteq \ell'}{\Gamma; \sigma \vdash e_1^\ell : t_1 \searrow M_1\,(c_{\ell'\,\ell}\,\sigma(\ell'))} \quad .$$

By the induction hypothesis, $\gamma(e_1) \approx\!\!\approx_\sigma \delta(M_1) : [t_1]_\ell$. By definition, there exist v and V such that $\gamma(e_1) \longrightarrow^* [v]_\ell$ and $\delta(M_1) \longrightarrow^* V$ and $[v]_\ell \leadsto_\sigma V : [t_1]_\ell$, and hence $v \approx\!\!\approx_\sigma V\,(c_{\ell'\,\ell}\,\sigma(\ell')) : [t_1]_\ell$. Since $\gamma(e_1^\ell)$ and $\delta(M_1\,(c_{\ell'\,\ell}\,\sigma(\ell')))$ respectively have the same normal forms as v and $V\,(c_{\ell'\,\ell}\,\sigma(\ell'))$, we conclude $\gamma(e_1^\ell) \approx\!\!\approx_\sigma \delta(M_1\,(c_{\ell'\,\ell}\,\sigma(\ell'))) : t_1$. $\qquad\square$

It is slightly harder to show that the logical correspondence includes the graphs of the inverse translation, since the inverse translation is *not* quite a (right) inverse of the translation to $\lambda^\rightarrow$: The inverse translation followed by the forward translation may yield a term different from the original (see Examples 4.4 and 4.10). Fortunately, the difference is only slight: They differ only in subterms of base types α_ℓ and are equivalent via $\equiv$, thus logically related by Lemma 3.9.

Lemma 5.5. *If* $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M : t^\dagger$ *and* $\Gamma; \sigma \vdash M \nearrow e : t$ *and* $\Gamma; \sigma \vdash e : t \searrow M'$, *then* $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M \equiv M' : t^\dagger$.

Proof. By induction on the derivation of $\Gamma; \sigma \vdash M \nearrow e : t$. We show only the main cases:

Case ($e = [e_1]_\ell$ and $\ell \notin dom(\sigma)$). Then, we can assume that the last steps of the translation and the inverse respectively have the following forms:

$$\frac{\Gamma; \sigma\{\ell \mapsto k\} \vdash M_1 \nearrow e_1 : t_1 \quad \ell \notin dom(\sigma)}{\Gamma; \sigma \vdash \lambda k{:}\alpha_\ell.\, M_1 \nearrow [e_1]_\ell : [t_1]_\ell}$$

$$\frac{\Gamma; \sigma\{\ell \mapsto k_0\} \vdash e_1 : t_1 \searrow M_2 \quad k_0\ \text{fresh}}{\Gamma; \sigma \vdash [e_1]_\ell : [t_1]_\ell \searrow \lambda k_0{:}\alpha_\ell.\, M_2} \quad .$$

By renaming the bound variables, we can also take k as k_0. Hence, by the induction hypothesis, $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger, k : \alpha_\ell \vdash M_1 \equiv M_2 : t_1^\dagger$, so $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash \lambda k{:}\alpha_\ell.\, M_1 \equiv \lambda k{:}\alpha_\ell.\, M_2 : \alpha_\ell \rightarrow t_1^\dagger$.

Case ($e = [e_1]_\ell$ and $\ell \in dom(\sigma)$). Then, we can assume that the last steps of the translation and the inverse respectively have the following forms:

$$\frac{\Gamma; \sigma\{\ell \mapsto k\} \vdash [k/\sigma(\ell)]M_1 \nearrow e_1 : t_1 \quad \ell \in dom(\sigma)}{\Gamma; \sigma \vdash \lambda k{:}\alpha_\ell.\, M_1 \nearrow [e_1]_\ell : [t_1]_\ell}$$

$$\frac{\Gamma; \sigma\{\ell \mapsto k_0\} \vdash e_1 : t_1 \searrow M_2 \quad k_0\ \text{fresh}}{\Gamma; \sigma \vdash [e_1]_\ell : [t_1]_\ell \searrow \lambda k_0{:}\alpha_\ell.\, M_2} \quad .$$

By renaming the bound variables, we can also take k as k_0. Hence, by the induction hypothesis, $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \backslash \{\sigma(\ell) : \alpha_\ell\}, \ k : \alpha_\ell \vdash [k/\sigma(\ell)]M_1 \equiv M_2 : t_1^\dagger$. Since $k = k_0$ and k_0 is fresh, $k \neq \sigma(\ell)$, so, by weakening, $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger, \ k : \alpha_\ell \vdash [k/\sigma(\ell)]M_1 \equiv M_2 : t_1^\dagger$. Applying Lemma 3.5 and the transitivity of $\equiv$, we have $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger, \ k : \alpha_\ell \vdash M_1 \equiv M_2 : t_1^\dagger$, and hence $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash \lambda k{:}\alpha_\ell.\, M_1 \equiv \lambda k{:}\alpha_\ell.\, M_2 : \alpha_\ell \to t_1^\dagger$.

Case $(e = e_1^\ell)$. Then, we can assume that the last steps of the translation and the inverse respectively have the following forms:

$$\frac{\Gamma; \sigma \vdash M_1 \nearrow e_1 : [t_1]_\ell \qquad \Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_0 : \alpha_\ell}{\Gamma; \sigma \vdash M_1\, M_0 \nearrow e_1^\ell : t_1}$$

$$\frac{\Gamma; \sigma \vdash e_1 : [t_1]_\ell \searrow M_2 \quad \ell' \in dom(\sigma) \quad \ell \sqsubseteq \ell'}{\Gamma; \sigma \vdash e_1^\ell : t_1 \searrow M_2\, (c_{\ell'\,\ell}\, \sigma(\ell'))} \quad .$$

Hence, by the induction hypothesis, $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_1 \equiv M_2 : \alpha_\ell \to t_1^\dagger$. Also, by definition, $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_0 \equiv c_{\ell'\,\ell}\, \sigma(\ell') : \alpha_\ell$. Hence $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M_1\, M_0 \equiv M_2\, (c_{\ell'\,\ell}\, \sigma(\ell')) : t_1^\dagger$. $\qquad\square$

Then, we can show the following theorem:

Theorem 5.6 (Inclusion of Inverse Translation). *If $\Gamma; \sigma \vdash M \nearrow e : t$ and $\gamma \approx\!\!\!\approx_\sigma \delta : \Gamma$, then $\gamma(e) \approx\!\!\!\approx_\sigma \delta(M) : t$.*

Proof. By Theorem 4.5, there exists M' such that $\Gamma; \sigma \vdash e : t \searrow M'$. Then, by Lemma 5.5, $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \vdash M \equiv M' : t^\dagger$. Since $C_\sqsubseteq, \sigma^\dagger \vdash \delta \approx \delta : \Gamma^\dagger$ (using Remark 3.10), $C_\sqsubseteq, \sigma^\dagger \vdash \delta(M) \approx \delta(M') : t^\dagger$ by Lemma 3.9. Then, by Theorem 5.4, $\gamma(e) \approx\!\!\!\approx_\sigma \delta(M') : t$ and, by Theorem 5.3 and the symmetricity of the logical relation for $\lambda^\to$, $\gamma(e) \approx\!\!\!\approx_\sigma \delta(M) : t$. $\qquad\square$

As a corollary, the logical correspondences is shown to be full.

Corollary 5.7 (Fullness of Logical Correspondences). *If $C_\sqsubseteq, \sigma^\dagger \vdash M : t^\dagger$, then there exists a $\lambda^{[]}$ term e such that $e \approx\!\!\!\approx_\sigma M : t$.*

Proof. By Theorem 3.1, there exists V such that $M \longrightarrow^* V$ and all the subderivations of $C_\sqsubseteq, \sigma^\dagger \vdash V : t^\dagger$ satisfy Subformula Property. Applying Theorem 4.8, we get the inverse e of V such that $\cdot; \sigma \vdash V \nearrow e : t$. So, from Theorem 5.6, $e \approx\!\!\!\approx_\sigma V : t$, and hence $e \approx\!\!\!\approx_\sigma M : t$. $\square$

5.2. **Preservation of Logical Relations.** By using the logical correspondence introduced above, we prove that the logical relations are preserved by the logical correspondence.

Theorem 5.8 (Preservation of Equivalences).
(1) *If $e_i \approx\!\!\!\approx_\sigma M_i : t$ for $i = 1, 2$ and $e_1 \approx_{dom(\sigma)} e_2 : t$, then $C_\sqsubseteq, \sigma^\dagger \vdash M_1 \approx M_2 : t^\dagger$.*
(2) *Symmetrically, if $e_i \approx\!\!\!\approx_\sigma M_i : t$ for $i = 1, 2$ and $C_\sqsubseteq, \sigma^\dagger \vdash M_1 \approx M_2 : t^\dagger$, then $e_1 \approx_{dom(\sigma)} e_2 : t$.*

Proof. We prove both simultaneously by induction on the structure of t. We show only the main cases:

Case $(t = t_1 \to t_2)$. To show (1), take arbitrary M_1' and M_2' such that $C_\sqsubseteq, \sigma^\dagger \vdash M_1' \approx M_2' : t_1^\dagger$. By fullness (Corollary 5.7), there exist e_i' such that $e_i' \approx\!\!\!\approx_\sigma M_i' : t_1$ $(i = 1, 2)$, and by the induction hypothesis (2) for t_1, we have $e_1' \approx_{dom(\sigma)} e_2' : t_1$. Then, by definition, there exist v_i and V_i such that $e_i \longrightarrow^* v_i$ and $M_i \longrightarrow^* V_i$ and $v_i\, e_i' \approx\!\!\!\approx_\sigma V_i\, M_i' : t_2$ for

$i = 1, 2$, and $v_1 \, e_1' \approx_{dom(\sigma)} v_2 \, e_2' : t_2$. Applying the induction hypothesis (1) for t_2 to them, $C_\sqsubseteq, \sigma^\dagger \ \vdash \ V_1 \, M_1' \approx V_2 \, M_2' : t_2^\dagger$. So we have $C_\sqsubseteq, \sigma^\dagger \ \vdash \ V_1 \sim V_2 : t_1^\dagger \to t_2^\dagger$, and hence $C_\sqsubseteq, \sigma^\dagger \ \vdash \ M_1 \approx M_2 : t_1^\dagger \to t_2^\dagger$. The statement (2) can be shown similarly, *without* the fullness.

Case $(t = [t_1]_\ell)$. To show (2), we have two subcases: $\ell \sqsubseteq dom(\sigma)$ or not. If $\ell \sqsubseteq \ell' \in dom(\sigma)$ for some ℓ', then, by definition, $C_\sqsubseteq, \sigma^\dagger \ \vdash \ c_{\ell' \, \ell} \, \sigma(\ell') \approx c_{\ell' \, \ell} \, \sigma(\ell') : \alpha_\ell$. Also, by definition, there exist v_i and V_i such that $e_i \longrightarrow^* [v_i]_\ell$ and $M_i \longrightarrow^* V_i$ and $v_i \approx\!\!\!>_\sigma V_i \, (c_{\ell' \, \ell} \, \sigma(\ell')) : t_1$ for $i = 1, 2$, and $C_\sqsubseteq, \sigma^\dagger \ \vdash \ V_1 \, (c_{\ell' \, \ell} \, \sigma(\ell')) \approx V_2 \, (c_{\ell' \, \ell} \, \sigma(\ell')) : t_1^\dagger$. Applying the induction hypothesis (2) for t_1, we have $v_1 \approx_{dom(\sigma)} v_2 : t_1$, which is equivalent to $v_1 \sim_{dom(\sigma)} v_2 : t_1$, so $e_1 \approx_{dom(\sigma)} e_2 : [t_1]_\ell$. The case $\ell \not\sqsubseteq dom(\sigma)$ is trivial. Showing (1) is easy since $C_\sqsubseteq, \sigma^\dagger \ \vdash M' : \alpha_\ell$ is equivalent to $\ell \sqsubseteq dom(\sigma)$. $\qquad\square$

5.3. **Noninterference.** Then, we prove the noninterference theorem by reducing it to Lemma 3.8.

Corollary 5.9 (Noninterference). *If* $\Gamma \, ; \, \pi \ \vdash \ e : t$ *and* $\gamma_1 \approx_\pi \gamma_2 : \Gamma$, *then* $\gamma_1(e) \approx_\pi \gamma_2(e) : t$.

Proof. Choose an arbitrary σ such that $dom(\sigma) = \pi$ and $ran(\sigma) \cap dom(\Gamma) = \emptyset$. By Theorem 4.5, $\Gamma; \sigma \vdash e : t \searrow M$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \ \vdash M : t^\dagger$ for some M. Similarly, for any $x \in dom(\gamma_i)$ $(i = 1, 2)$, there exists M_{xi} such that $\cdot; \sigma \ \vdash \ \gamma_i(x) : \Gamma(x) \searrow M_{xi}$ and $\Gamma^\dagger, C_\sqsubseteq, \sigma^\dagger \ \vdash \ M_{xi} : (\Gamma(x))^\dagger$. Define δ_i $(i = 1, 2)$ as a simultaneous substitution such that $dom(\delta_i) = dom(\gamma_i)$ and $\delta_i(x) = M_{xi}$ for $x \in dom(\delta_i)$. Then, by Theorem 5.4, $\gamma_i \approx\!\!\!>_\sigma \delta_i : \Gamma$ for $i = 1, 2$ and so $\gamma_i(e) \approx\!\!\!>_\sigma \delta_i(M) : t$ for $i = 1, 2$. By applying Theorem 5.8(1) to the assumption $\gamma_1 \approx_\pi \gamma_2 : \Gamma$, we have $C_\sqsubseteq, \sigma^\dagger \ \vdash \ \delta_1 \approx \delta_2 : \Gamma^\dagger$. Thus, by Lemma 3.8 (with Remark 3.10), $C_\sqsubseteq, \sigma^\dagger \ \vdash \ \delta_1(M) \approx \delta_2(M) : t^\dagger$. Finally, by Theorem 5.8(2), $\gamma_1(e) \approx_\pi \gamma_2(e) : t$. $\qquad\square$

6. Comparison of DCC with $\lambda^{[]}$

In this section, we briefly review DCC [1] and discuss why the translation from DCC to System F given by Tse and Zdancewic [22, 23] is neither full nor even sound. Then, we discuss an extension DCC_{pc} of DCC, which was proposed also by Tse and Zdancewic in order to make the translation full [21, 22, 23]. Finally, we show that DCC_{pc} is equivalent to $\lambda^{[]}$ by giving translations between the two.

6.1. **DCC and Tse–Zdancewic's translation to System F.** DCC is an extension of the computational λ-calculus [12] and uses monads indexed by dependency levels (e.g., security levels, binding times) in order to control the dependencies between computations. The dependency levels are partially ordered by $\sqsubseteq^2$ as in $\lambda^{[]}$; computation and data at a higher level are permitted to depend on those at lower levels, but the other direction of dependencies is forbidden. Here, we briefly sketch a simplified version of DCC [22, 23] (we call it simply DCC), in which pointed types and recursion are omitted.

---

[2] In fact, the dependency levels were assumed be a lattice [1] but we do not need meets and joins in the following development.

The syntax of DCC is defined as follows:

$$t ::= unit \mid t \rightarrow t \mid t \times t \mid t + t \mid T_\ell \ t$$

$$e ::= x \mid () \mid \lambda x{:}t.\, e \mid e\, e \mid \langle e,\, e \rangle \mid \pi_1(e) \mid \pi_2(e) \mid \iota_1(e) \mid \iota_2(e)$$

$$\mid (\mathbf{case}\, e\, \mathbf{of}\, \iota_1(x_1).e \mid \iota_2(x_2).e) \mid \eta_\ell\, e \mid \mathtt{bind}\ x\ =\ e\ \mathtt{in}\ e$$

Roughly speaking, a monadic type $T_\ell\ t$, the monadic unit $\eta_\ell\ e$, and the bind operation $\mathtt{bind}\ x = e_1\ \mathtt{in}\ e_2$ correspond to sealing types $[t]_\ell$, sealing terms $[e]_\ell$, and unsealing terms e^ℓ, respectively. The typing rule for η_ℓ is as follows:

$$\frac{\Gamma \vdash e : t}{\Gamma \vdash \eta_\ell\, e : T_\ell\ t}$$

Note that a type judgment of DCC lacks an observer level; instead, the notion of protected types is introduced to prevent information leakage and plays a key role in the following typing rule for $\mathtt{bind}$:

$$\frac{\Gamma \vdash e_1 : T_\ell\ t_1 \qquad \Gamma, x : t_1 \vdash e_2 : t_2 \qquad \ell \preceq t_2}{\Gamma \vdash \mathtt{bind}\ x = e_1\ \mathtt{in}\ e_2 : t_2}$$

$$\ell \preceq\ unit \qquad \frac{\ell \preceq t_1 \quad \ell \preceq t_2}{\ell \preceq t_1 \times t_2} \qquad \frac{\ell \preceq t_2}{\ell \preceq t_1 \rightarrow t_2} \qquad \frac{\ell \not\sqsubseteq \ell' \quad \ell \preceq t}{\ell \preceq T_{\ell'}\ t} \qquad \frac{\ell \sqsubseteq \ell'}{\ell \preceq T_{\ell'}\ t}$$

Here, judgment $\ell \preceq t$ is read as "t is protected at ℓ". Intuitively, this judgment means that observers only at a level equal to or higher than ℓ can obtain some bits of information from the value of t.

So, this rule ensures that the value of the whole term cannot be examined at unrelated levels. However, $\mathtt{bind}$ is restrictive in the sense that η_ℓ must be placed within the scope of x to make t_2 protected. For example, the term $\lambda y : T_\ell\ \mathtt{bool}.\mathtt{bind}\ x = y\ \mathtt{in}\ \eta_\ell\, x$ is given type $(T_\ell\ \mathtt{bool}) \rightarrow (T_\ell\ \mathtt{bool})$ while the term $\lambda y : T_\ell\ \mathtt{bool}.\eta_\ell\, (\mathtt{bind}\ x = y\ \mathtt{in}\ x)$ cannot. We will see that this restriction is a source of the failure of fullness of the translation by Tse and Zdancewic. The other typing rules are the same as $\lambda^\rightarrow$.

The reduction rule for $\mathtt{bind}$ is $\mathtt{bind}\ x = \eta_\ell\, e_1\ \mathtt{in}\ e_2 \longrightarrow [e_1/x]e_2$. The other reduction rules and the logical relations are essentially the same as $\lambda^{[]}$ except for the change from $[t]_\ell$ to $T_\ell\, t$. The logical relations are indexed by an observer level (that is, a finite set of data levels) rather than a single data level as in Tse and Zdancewic [22, 23, 21]. Although our definition is a straightforward extension of theirs, this seems more natural for $\mathrm{DCC_{pc}}$ below, for the domains of the relations are terms that are well typed at a given observer level.

A main idea of the translation by Tse and Zdancewic, which we have followed in this paper, is to translate monadic types $T_\ell\ t$ into function types $\alpha_\ell \rightarrow t$. (Otherwise, type translation is the same as ours.) Term translation, the details for which we refer to [22, 23], is more involved than our translation, due to the complexity of $\mathtt{bind}$ and protected types— we will see how they are expressed in terms of our unsealing in the next section.

6.2. **Failure of Fullness and Soundness.** Now we explain why their translation is neither full nor sound.

Consider the DCC type $t = T_\ell((T_\ell \text{ bool}) \to \text{bool})$. Then, any DCC terms of this type is equivalent to (sealed) constant functions $\eta_\ell(\lambda x : T_\ell \text{ bool}.c)$ where c is either true or false. Note, in particular, that the term $e = \eta_\ell(\lambda y : T_\ell \text{ bool}.\text{bind } x = y \text{ in } x)$ is *ill typed* due to the restriction of the typing rule of bind. As a result, the two terms

$$e_1 = \lambda f.\text{bind } f' = f \text{ in } \eta_\ell (f' (\eta_\ell \text{ true}))$$

and

$$e_2 = \lambda f.\text{bind } f' = f \text{ in } \eta_\ell (f' (\eta_\ell \text{ false}))$$

are logically related at the type $(T_\ell((T_\ell \text{ bool}) \to \text{bool})) \to (T_\ell \text{ bool})$ and level ℓ since all we can pass to these functions are the constant functions above and we cannot pass non-constant functions such as e.

In System F, however, the translations of e_1 and e_2 are *not* logically related at type $\alpha_\ell \to ((\alpha_\ell \to \text{bool}) \to \text{bool})$, which corresponds to the DCC type t above! This is because they can be distinguished by applying them to the term $M = \lambda k : \alpha_\ell.\lambda f : \alpha_\ell \to \text{bool}.fk$, which would correspond to e.

In short, there is no well typed DCC term that corresponds to M (failure of fullness) and, as a result, the equivalence of e_1 and e_2 is not preserved through the translation (failure of soundness).

6.3. **Tse and Zdancewic's Extension of DCC.** Interestingly, Tse and Zdancewic also noticed the restriction of the typing for bind in DCC and proposed an extension of DCC by introducing the notion of protection contexts (as a set of data levels) to type judgments. The typing rules for η_ℓ and bind are changed as follows:

$$\frac{\Gamma\,;\, \pi \cup \{\ell\} \vdash e : t}{\Gamma\,;\, \pi \vdash \eta_\ell e : T_\ell t} \tag{D-Eta}$$

$$\frac{\Gamma\,;\, \pi \vdash e : T_\ell t \quad \Gamma, x : t\,;\, \pi \vdash e' : t' \quad \ell \sqsubseteq \pi}{\Gamma\,;\, \pi \vdash \text{bind } x = e \text{ in } e' : t'} \tag{D-Bind1}$$

$$\frac{\Gamma\,;\, \pi \vdash e : T_\ell t \quad \Gamma, x : t\,;\, \pi \vdash e' : t' \quad \ell \not\sqsubseteq \pi \quad \ell \preceq t'}{\Gamma\,;\, \pi \vdash \text{bind } x = e \text{ in } e' : t'} \tag{D-Bind2}$$

$$\frac{\Gamma\,;\, \pi \cup \{\ell\} \vdash e : t \quad \ell \not\sqsubseteq \pi \quad \ell \preceq t}{\Gamma\,;\, \pi \vdash e : t} \tag{D-Protected}$$

The rule (D-Bind1) is essential and just corresponds to the rule (ST-Unseal) of $\lambda^{[]}$. The rule (D-Protected) means that a term of a type protected by ℓ can be used by a user which does not have ℓ. This extension allows terms like $\lambda y : T_\ell \text{ bool}.\eta_\ell(\text{bind } x = y \text{ in } x)$ and $\eta_\ell(\lambda y : T_\ell \text{ bool}.\text{bind } x = y \text{ in } x)$ to be well typed. The rest of the typing rules are the same as $\lambda^{[]}$. The definitions of the reduction rules and the logical relations are the same as DCC.

In the next subsection, we will show the three rules (D-Bind1), (D-Bind2), and (D-Protected) are in fact derived forms in the sense that DCC_{pc} and $\lambda^{[]}$ are equivalent.

Remark 6.1. $\mathrm{DCC_{pc}}$ was proposed [22, 23] and simplified later by Tse and Zdancewic [21]. In this paper, we use the simplified version with the following changes:

- We split the single typing rule for `bind` into the two rules.
- We add the rule (D-Protected) above for the subject reduction property, which does not really hold in the original formulation, due to the reduction of `bind`.

6.4. **Isomorphisms between $\lambda^{[]}$ and $\mathrm{DCC_{pc}}$.** We show correspondence between $\lambda^{[]}$ and $\mathrm{DCC_{pc}}$ by giving a translation $(\cdot)^{\bullet}$ from $\lambda^{[]}$ to $\mathrm{DCC_{pc}}$ and its inverse $(\cdot)^{\circ}$ and showing that both preserve logical equivalences. The inverse translation is inspired by Tse and Zdancewic's translation from DCC to System F [22, 23]: We obtain the inverse translation by comparing theirs with our full complete translation from $\lambda^{[]}$ to $\lambda^{\rightarrow}$. In what follows, we add subscripts "$\lambda^{[]}$" and "$\mathrm{DCC_{pc}}$" to distinguish typing judgments of the two calculi.

At the type level, both translations are easy—they just exchange $[\cdot]_{\ell}$ and T_{ℓ}:

$$([t]_{\ell})^{\bullet} \overset{\mathrm{def}}{=} T_{\ell} \, (t^{\bullet}) \qquad (T_{\ell} \, t)^{\circ} \overset{\mathrm{def}}{=} [t^{\circ}]_{\ell}$$

(For other type constructors, both translations are trivial.) At the term level, $(\cdot)^{\bullet}$ is obvious—sealing and unsealing can be straightforwardly expressed by η_{ℓ} and `bind`, respectively:

$$([e]_{\ell})^{\bullet} \overset{\mathrm{def}}{=} \eta_{\ell} \, (e^{\bullet})$$

$$(e^{\ell})^{\bullet} \overset{\mathrm{def}}{=} \texttt{bind } x = e^{\bullet} \texttt{ in } x.$$

The translation $(\cdot)^{\circ}$ for terms is more involved. A main difficulty is in the `bind` operator. At first one might think `bind` $x = e_1$ `in` e_2 can be expressed by $(\lambda x.e_2^{\circ}) \, (e_1^{\circ})^{\ell}$, but, if $\Gamma; \pi \vdash_{\mathrm{DCC_{pc}}} \texttt{bind } x = e_1 \texttt{ in } e_2 : t_2$ is derived by (D-Bind2), where $\ell \not\sqsubseteq \pi$ and $\ell \preceq t_2$, then $(e_1^{\circ})^{\ell}$ is typable only at $\pi \cup \{\ell\}$, which is *strictly* higher than π; so is $(\lambda x.e_2^{\circ}) \, (e_1^{\circ})^{\ell}$. Thus, this naive translation does not quite preserve typing.

This problem is solved by observing that t_2 is protected at ℓ (i.e., $\ell \preceq t_2$). First, we can seal $(\lambda x.e_2^{\circ}) \, (e_1^{\circ})^{\ell}$ and derive $\Gamma^{\circ}; \pi \vdash_{\lambda^{[]}} [(\lambda x.e_2^{\circ}) \, (e_1^{\circ})^{\ell}]_{\ell} : [t_2^{\circ}]_{\ell}$. Here, this sealing with ℓ is redundant since t_2 is already protected by ℓ. In fact, we can always eliminate such a sealing by applying an *anti-protection combinator*, defined below, of type $[t_2]_{\ell} \rightarrow t_2$.

Definition 6.2 (Anti-Protection Combinators). The set of closed terms $\mathbb{P}_{\ell \preceq t}$ indexed by protected types is inductively defined as follows:

$$\mathbb{P}_{\ell \preceq unit} = \lambda x : [unit]_{\ell}. \, ()$$

$$\mathbb{P}_{\ell \preceq t_1 \times t_2} = \lambda x : [t_1 \times t_2]_{\ell}. \, \langle \mathbb{P}_{\ell \preceq t_1}[\pi_1(x^{\ell})]_{\ell}, \mathbb{P}_{\ell \preceq t_2}[\pi_2(x^{\ell})]_{\ell} \rangle$$

$$\mathbb{P}_{\ell \preceq t_1 \rightarrow t_2} = \lambda x : [t_1 \rightarrow t_2]_{\ell}. \, \lambda y : t_1. \, \mathbb{P}_{\ell \preceq t_2} \, [x^{\ell} \, y]_{\ell}$$

$$\mathbb{P}_{\ell \preceq T_{\ell'} \, t} = \lambda x : [[t]_{\ell'}]_{\ell}. \, [(x^{\ell})^{\ell'}]_{\ell'} \qquad\qquad \text{if} \quad \ell \sqsubseteq \ell'$$

$$\mathbb{P}_{\ell \preceq T_{\ell'} \, t} = \lambda x : [[t]_{\ell'}]_{\ell}. \, [\mathbb{P}_{\ell \preceq t} \, [(x^{\ell})^{\ell'}]_{\ell}]_{\ell'} \quad \text{if} \quad \ell \not\sqsubseteq \ell' \quad \text{and} \quad \ell \preceq t$$

These combinators intuitively mean that, for any $\lambda^{[]}$ term e of type t° such that $\ell \preceq t$, the sealing term $[e]_{\ell}$ can be unsealed at any observer level. This intuition is justified by the following proposition:

Proposition 6.3. *The following properties hold:*

(1) *If $\ell \preceq t$ and $\ell \sqsubseteq \pi$, then $\mathbb{P}_{\ell \preceq t} \approx_\pi \lambda x \!:\! [t^\circ]_\ell . \, x^\ell : [t^\circ]_\ell \to t^\circ$.*
(2) *If $\ell \preceq t$ and $\ell \not\sqsubseteq \pi$, then $e_1 \approx_\pi e_2 : t^\circ$ for any $\lambda^{[]}$ terms e_i such that $\cdot \, ; \pi \vdash_{\lambda^{[]}} e_i : t^\circ$ $(i = 1, 2)$. In particular, under the same assumptions, it follows that $\mathbb{P}_{\ell \preceq t} \approx_\pi f : [t^\circ]_\ell \to t^\circ$ for any function f such that $\cdot \, ; \pi \vdash_{\lambda^{[]}} f : [t^\circ]_\ell \to t^\circ$.*

Proof. By induction of the derivation of $\ell \preceq t$. $\square$

The second clause means that no term of a protected type illegally leak any information. A corresponding property has been proved for DCC [1].

Now we return to defining $(\cdot)^\circ$. For the `bind` operator, we have two cases. (Strictly speaking, $(\cdot)^\circ$ is defined by induction on the type derivation as in Section 4.) If the last typing rule is (D-BIND1), the definition is just

$$(\texttt{bind } x = e_1 \texttt{ in } e_2)^\circ \stackrel{\text{def}}{=} (\lambda x. \, e_2^\circ) \, (e_1^\circ)^\ell,$$

where e_1 and e_2 have types $T_\ell \, t_1$ and t_2, respectively. If it is (D-BIND2), we can assume $\ell \preceq t_2$ and

$$(\texttt{bind } x = e_1 \texttt{ in } e_2)^\circ \stackrel{\text{def}}{=} \mathbb{P}_{\ell \preceq t_2} \, [(\lambda x. \, e_2^\circ) \, (e_1^\circ)^\ell]_\ell.$$

Another interesting case is when the last step of the type derivation is

$$\frac{\Gamma \, ; \pi \cup \{\ell\} \vdash_{\mathrm{DCC_{pc}}} e : t \qquad \ell \not\sqsubseteq \pi \qquad \ell \preceq t}{\Gamma \, ; \pi \vdash_{\mathrm{DCC_{pc}}} e : t} \qquad \text{(D-PROTECTED)}$$

The situation is similar to the case for (D-BIND2): the $\mathrm{DCC_{pc}}$ type t is already protected at ℓ and so ℓ in the context of the premise is redundant. So, we obtain $\mathbb{P}_{\ell \preceq t} \, [e^\circ]_\ell$, in which e° is the translation from $\Gamma \, ; \pi \cup \{\ell\} \vdash_{\mathrm{DCC_{pc}}} e : t$. For the other typing rules, the translation is trivial. For example,

$$(\eta_\ell \, e)^\circ \stackrel{\text{def}}{=} [e^\circ]_\ell.$$

Clearly, both translations preserve typing. The following theorem ensures that the translations preserve the logical relations, showing $\mathrm{DCC_{pc}}$ and $\lambda^{[]}$ are equivalent.

Theorem 6.4 (Preservation of Equivalences). *$e_1 \approx_\pi e_2 : t$ in $\mathrm{DCC_{pc}}$ iff $e_1^\circ \approx_\pi e_2^\circ : t^\circ$ in $\lambda^{[]}$. Also, $e_1^\bullet \approx_\pi e_2^\bullet : t^\bullet$ in $\mathrm{DCC_{pc}}$ iff $e_1 \approx_\pi e_2 : t$ in $\lambda^{[]}$.*

Proof. We just give a sketch, which is along a similar line as the proof of Theorem 5.8. First, like Definition 5.1, we define logical correspondences $e \approx\!\!\!\approx_\pi e' : t$ over terms of $\lambda^{[]}$ and $\mathrm{DCC_{pc}}$ indexed by observer levels π (instead of finite maps, since both $\lambda^{[]}$ and $\mathrm{DCC_{pc}}$ use the common poset of data levels). Then we show the inclusion of $(\cdot)^\circ$ and $(\cdot)^\bullet$, that is, $e \approx\!\!\!\approx_\pi e^\circ : t$ and $e^\bullet \approx\!\!\!\approx_\pi e : t$ (cf. Theorem 5.4 and 5.6). We use Proposition 6.3 to prove the former. Finally, we show the preservation of the equivalences (cf. Theorem 5.8) and, combining the inclusion of the translations, get the result. $\square$
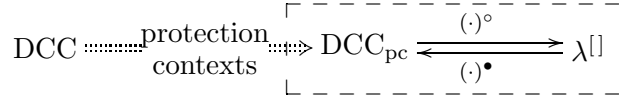
$$\text{DCC} \xdashrightarrow{\text{protection contexts}} \text{DCC}_{\text{pc}} \underset{(\cdot)^{\bullet}}{\overset{(\cdot)^{\circ}}{\rightleftarrows}} \lambda^{[]}$$

Figure 1: Relationship among DCC, DCC_{pc}, and $\lambda^{[]}$.

7. CONCLUSION

We have formalized noninterference for a typed λ-calculus $\lambda^{[]}$ by logical relations and proved it by reducing it to the basic lemma of logical relation for $\lambda^{\rightarrow}$ through a translation of $\lambda^{[]}$ to $\lambda^{\rightarrow}$. Moreover, we have shown that $\lambda^{[]}$ is equivalent to DCC_{pc}, an extension of DCC with observer levels, as illustrated in Figure 1: a dotted double arrow stands for a language extension and the two systems (except DCC) in the dashed box have sound and fully complete translations into $\lambda^{\rightarrow}$. In those systems, dependency is captured by typability in $\lambda^{\rightarrow}$ through the translations.

There have been presented many ways to prove noninterference theorems for type-based dependency analyses for higher-order languages. For example, Heintze and Riecke [7] and Abadi et al. [1] showed the noninterference theorem for SLam by using denotational semantics. Pottier and Simonet [15] proved it for Core ML with non-standard operational semantics. Miyamoto and Igarashi [10], in the study of a modal typed calculus $\lambda_s^{\square}$, showed that the noninterference theorem for certain types can be easily proved only by using a simple nondeterministic reduction system, although this system does not include recursion unlike the others mentioned here.

In comparison with these proofs, the proof technique presented in this paper might seem overwhelming to show only noninterference. Nevertheless, we believe it is still theoretically interesting since the translation shows that the notion of dependency can be captured only in terms of simple types and makes a comparison between type-based dependency analyses easier.

Practically, the translation might be a basis for implementing a language with sealing by another language without it. However, our results rely on full reduction with commuting conversions, or strong normalization, which cannot be assumed in real languages. So, it would be interesting future work to investigate how this proof technique may be extended to richer languages with, for example, recursion. To add recursion, several difficulties have to be overcome. A first problem, as is already pointed out by Tse and Zdancewic [21, 22, 23], is that a key of any data level can be "forged" by using recursion, which allows a term of any type, and such forged keys enable any observer to extract a sealed value illegally. As suggested also by Tse and Zdancewic, this problem may be solved by pointed types (or use of Haskell's `seq`). A second, more serious problem is that it would be much harder to give an inverse translation: if the translation is extended in a straightforward manner, then there will be "junk" terms, such as some divergent terms not in the image of the translation and, as a result, fullness would be lost. We expect some more significant work will be needed to solve these problems.

Acknowledgements

References

[1] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *POPL '99: Proceedings of 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 147–160, New York, NY, USA, 1999. ACM Press.

[2] Philippe de Groote. On the strong normalisation of intuitionistic natural deduction with permutative-conversions. *Information and Computation*, 178:441–464, August 2002.

[3] Dorothy. E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, July 1977.

[4] Jean-Yves Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. PhD thesis, Université Paris VII, 1972. A summary appeared in the Proceedings of the Second Scandinavian Logic Symposium (J.E. Fenstad, editor), North-Holland, 1971 (pp. 63–92).

[5] Joseph Goguen and José Meseguer. Security policies and security models. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[6] Masahito Hasegawa. Girard translation and logical predicates. *Journal of Functional Programming*, 10(1):77–89, January 2000.

[7] Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In *POPL '98: Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 365–377, 1998.

[8] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, 1993.

[9] John C. Mitchell. *Foundations for Programming Languages*. The MIT Press, 1996.

[10] Kenji Miyamoto and Atsushi Igarashi. A modal foundation for secure information flow. In *FCS '04: Proceedings of Workshop on Foundations of Computer Security*, pages 187–203, June 2004.

[11] Masaaki Mizuno and David A. Schmidt. A security flow control algorithm and its denotational semantics correctness proof. *Formal Aspects of Computing*, 4(6A):727–754, 1992.

[12] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 1:55–92, 1991.

[13] Maxwell H. A. Newman. On theories with a combinatorial definition of "equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[14] Gordon D. Plotkin. Lambda-definability in the full type hierarchy. In *To H.B.Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[15] François Pottier and Vincent Simonet. Information flow inference for ML. *ACM Transactions on Programming Languages and Systems*, 25(1):117–158, 2003.

[16] John C. Reynolds. Towards a theory of type structure. In *Proc. Colloque sur la Programmation*, pages 408–425, New York, 1974. Springer-Verlag LNCS 19.

[17] John C. Reynolds. Types, abstraction and parametric polymorphism. In *Proceedings of the IFIP 9th World Computer Congress*, pages 513–523, 1983.

[18] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal On Selected Areas In Communications*, 21(1):5–19, 2003.

[19] Naokata Shikuma and Atsushi Igarashi. Proving noninterference by a fully complete translation to the simply typed λ-calculus. In *ASIAN '06: Proceedings of the 11th Annual Asian Computing Science Conference*, volume 4435 of *LNCS*, pages 302–316. Springer-Verlag, December 2006.

[20] Yan Mei Tang and Pierre Jouvelot. Effect systems with subtyping. In *Proceedings of ACM Symposium on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'95)*, pages 45–53, 1995.

[21] Stephen Tse and Steve Zdancewic. Translating dependency into parametricity. A draft accepted by Journal of Functional Programming (JFP), January 2006. (Submitted, December 2004.) Available as `http://www.cis.upenn.edu/~stevez/stse-work/dccsysf/jfp.pdf`.

[22] Stephen Tse and Steve Zdancewic. Translating dependency into parametricity. In *ICFP '04: Proceedings of 9th ACM International Conference on Functional Programming*, pages 115–125, New York, NY, USA, 2004. ACM Press.

[23] Stephen Tse and Steve Zdancewic. Translating dependency into parametricity. Technical Report MIS-CIS-04-01, University of Pennsylvania, 2004. Extended version of [22].

[24] Philip Wadler. Theorems for free! In *FPCA '89: Proceedings of the 4th International Conference on Functional Programming Languages and Computer Architecture*, pages 347–359. ACM, New York, NY, USA, 1989.