

Type-based Declassification for Free (with appendix)

Minh Ngo^{1,2}, David A. Naumann¹, and Tamara Rezk²

¹ Stevens Institute of Technology

² Inria

Abstract. This work provides a study to demonstrate the potential of using off-the-shelf programming languages and their theories to build sound language-based-security tools. Our study focuses on information flow security encompassing declassification policies that allow us to express flexible security policies needed for practical requirements. We translate security policies, with declassification, into an interface for which an unmodified standard typechecker can be applied to a source program—if the program typechecks, it provably satisfies the policy. Our proof reduces security soundness—with declassification—to the mathematical foundation of data abstraction, Reynolds’ abstraction theorem.

To appear in ICFEM 2020

1 Introduction

A longstanding challenge for software systems is the enforcement of security in applications implemented in conventional general-purpose programming languages. For high assurance, precise mathematical definitions are needed for policies, enforcement mechanism, and program semantics. The latter, in particular, is a major challenge for languages in practical use. In order to minimize the cost of assurance, especially over time as systems evolve, it is desirable to leverage work on formal modeling with other goals such as functional verification, equivalence checking, and compilation.

To be auditable by stakeholders, policy should be expressed in an accessible way. This is one of several reasons why types play an important role in many works on information flow (IF) security. For example, Flowcaml [36] and Jif [29] express policy using types that include IF labels. They statically enforce policy using dedicated IF type checking and inference. Techniques from type theory are also used in security proofs such as those for Flowcaml and the calculus DCC [1].

IF is typically formalized as the preservation of indistinguishability relations between executions. Researchers have hypothesized that this should be an instance of a celebrated semantics basis in type theory: relational parametricity [39]. Relational parametricity provides an effective basis for formal reasoning about program transformations (“theorems for free” [53]), representation independence and information hiding for program verification [28,6]. The connection between IF and relational parametricity has been made precise in 2015, for DCC,

by translation to the calculus F_ω and use of the existing parametricity theorem for F_ω [12]. The connection is also made, perhaps more transparently, in a translation of DCC to dependent type theory, specifically the calculus of constructions and its parametricity theorem [4].

In this work, we advance the state of the art in the connection between IF and relational parametricity, guided by three main goals. One of the goals motivating our work is to *reduce the burden of defining dedicated type checking, inference, and security proofs* for high assurance in programming languages. A promising approach towards this goal is the idea of leveraging type abstraction to enforce policy, and in particular, *leveraging the parametricity theorem to obtain security guarantees*. A concomitant goal is *to do so for practical IF policies* that encompass selective declassification, which is needed for most policies in practice. For example, a password checker program or a program that calculates aggregate or statistical information must be considered insecure without declassification.

To build on the type system and theory of a language without *a priori* IF features, policy needs to be encoded somehow, and the program may need to be transformed. For example, to prove that a typechecked DCC term is secure with respect to the policy expressed by its type, Bowman and Ahmed [12] encode the typechecking judgment by nontrivial translation of both types and terms into F_ω . Any translation becomes part of the assurance argument. Most likely, complicated translation will also make it more difficult to use extant type checking/inference (and other development tools) in diagnosing security errors and developing secure code. This leads us to highlight a third goal, needed to achieve the first goal, namely to *minimize the complexity of translation*.

There is a major impediment to leveraging type abstraction: few languages are relationally parametric or have parametricity theorems. The lack of parametricity can be addressed by focusing on well behaved subsets and leveraging additional features like ownership types that may be available for other purposes (e.g., in the Rust language). As for the paucity of parametricity theorems, we take hope in the recent advances in machine-checked metatheory, such as correctness of the CakeML and CompCert compilers, the VST logic for C, the relational logic of Iris. For parametricity specifically, the most relevant work is Cray’s formal proof of parametricity for the ML module calculus [14].

Contributions. Our *first contribution* is to translate policies with declassification—in the style of relaxed noninterference [27]—into abstract types in a functional language, in such a way that typechecking the original program implies its security. For doing so, we neither rely on a specialized security type system [12] nor on modifications of existing type systems [17]. A program that typechecks may use the secret inputs parametrically, e.g., storing in data structures, but cannot look at the data until declassification has been applied. Our *second contribution* is to prove security by direct application of a parametricity theorem. We carry out this development for the polymorphic lambda calculus, using the original theorem of Reynolds. We also provide an appendix that shows this development for the ML module calculus using Cray’s theorem [14], enabling the use of ML to check security.

2 Background: Language and Abstraction Theorem

To present our results we choose the simply typed and call-by-value lambda calculus, with integers and type variables, for two reasons: (1) the chosen language is similar to the language used in the paper of Reynolds [39] where the abstraction theorem was first proven, and (2) we want to illustrate our encoding approach (§4) in a minimal calculus. This section defines the language we use and recalls the abstraction theorem, a.k.a. parametricity. Our language is very close to the one in Reynolds [39, § 2]; we prove the abstraction theorem using contemporary notation.³

Language. The syntax of the language is as below, where α denotes a type variable, x a term variable, and n an integer value. A value is *closed* when there is no free term variable in it. A type is *closed* when there is no type variable in it.

| | |
|--|----------------|
| $\tau ::= \mathbf{int} \mid \alpha \mid \tau_1 \times \tau_2 \mid \tau_1 \rightarrow \tau_2$ | Types |
| $v ::= n \mid \langle v, v \rangle \mid \lambda x : \tau. e$ | Values |
| $e ::= x \mid v \mid \langle e, e \rangle \mid \pi_i e \mid e_1 e_2$ | Terms |
| $E ::= [\cdot] \mid \langle E, e \rangle \mid \langle v, E \rangle \mid \pi_i E \mid E e \mid v E$ | Eval. Contexts |

We use small-step semantics, with the reduction relation \rightarrow defined inductively by these rules.

$$\pi_i \langle v_1, v_2 \rangle \rightarrow v_i \qquad (\lambda x : \tau. e) v \rightarrow e[x \mapsto v] \qquad \frac{e \rightarrow e'}{E[e] \rightarrow E[e']}$$

We write $e[x \mapsto e']$ for capture-avoiding substitution of e' for free occurrences of x in e . We use parentheses to disambiguate term structure and write \rightarrow^* for the reflexive, transitive closure of \rightarrow .

A *typing context* Δ is a set of type variables. A *term context* Γ is a mapping from term variables to types, written like $x : \mathbf{int}, y : \mathbf{int} \rightarrow \mathbf{int}$. We write $\Delta \vdash \tau$ to mean that τ is *well-formed w.r.t.* Δ , that is, all type variables in τ are in Δ . We say that e is *typable w.r.t.* Δ and Γ (denoted by $\Delta, \Gamma \vdash e$) when there exists a well-formed type τ such that $\Delta, \Gamma \vdash e : \tau$. The derivable typing judgments are defined inductively in Fig. 1. The rules are to be instantiated only with Γ that is well-formed under Δ , in the sense that $\Delta \vdash \Gamma(x)$ for all $x \in \text{dom}(\Gamma)$. When the term context and the type context are empty, we write $\vdash e : \tau$.

Logical relation. The logical relation is a type-indexed family of relations on values, parameterized by given relations for type variables. From it, we derive a relation on terms. The abstraction theorem says the latter is reflexive.

³ Some readers may find it helpful to consult the following references for background on logical relations and parametricity: [25, Chapt. 49], [28, Chapt. 8], [13], [34].

$$\begin{array}{c}
\text{FT-INT} \frac{}{\Delta, \Gamma \vdash n : \mathbf{int}} \qquad \text{FT-VAR} \frac{x : \tau \in \Gamma}{\Delta, \Gamma \vdash x : \tau} \\
\text{FT-PAIR} \frac{\Delta, \Gamma \vdash e_1 : \tau_1 \quad \Delta, \Gamma \vdash e_2 : \tau_2}{\Delta, \Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \qquad \text{FT-PRJ} \frac{\Delta, \Gamma \vdash e : \tau_1 \times \tau_2}{\Delta, \Gamma \vdash \pi_i e : \tau_i} \\
\text{FT-FUN} \frac{\Delta, \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta, \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \\
\text{FT-APP} \frac{\Delta, \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Delta, \Gamma \vdash e_2 : \tau_1}{\Delta, \Gamma \vdash e_1 e_2 : \tau_2}
\end{array}$$

Fig. 1. Typing rules

Let γ be a *term substitution*, i.e., a finite map from term variables to closed values, and δ be a *type substitution*, i.e., a finite map from type variables to closed types. In symbols:

$$\begin{array}{ll}
\gamma ::= . \mid \gamma, x \mapsto v & \text{Term Substitutions} \\
\delta ::= . \mid \delta, \alpha \mapsto \tau, \text{ where } \vdash \tau & \text{Type Substitutions}
\end{array}$$

We say γ *respects* Γ (denoted by $\gamma \models \Gamma$) when $\text{dom}(\gamma) = \text{dom}(\Gamma)$ and $\vdash \gamma(x) : \Gamma(x)$ for any x . We say δ *respects* Δ (denoted by $\delta \models \Delta$) when $\text{dom}(\delta) = \Delta$. Let $\text{Rel}(\tau_1, \tau_2)$ be the set of all binary relations over closed values of closed types τ_1 and τ_2 . Let ρ be an *environment*, a mapping from type variables to relations $R \in \text{Rel}(\tau_1, \tau_2)$. We write $\rho \in \text{Rel}(\delta_1, \delta_2)$ to say that ρ is compatible with δ_1, δ_2 as follows: $\rho \in \text{Rel}(\delta_1, \delta_2) \triangleq \text{dom}(\rho) = \text{dom}(\delta_1) = \text{dom}(\delta_2) \wedge \forall \alpha \in \text{dom}(\rho). \rho(\alpha) \in \text{Rel}(\delta_1(\alpha), \delta_2(\alpha))$. The logical relation is inductively defined in Fig. 2, where $\rho \in \text{Rel}(\delta_1, \delta_2)$ for some δ_1 and δ_2 . For any τ , $\llbracket \tau \rrbracket_\rho$ is a relation on closed values. In addition, $\llbracket \tau \rrbracket_\rho^{\text{ev}}$ is a relation on terms.

Lemma 1. *Suppose that $\rho \in \text{Rel}(\delta_1, \delta_2)$ for some δ_1 and δ_2 . For $i \in \{1, 2\}$, it follows that:*

- if $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_\rho$, then $\vdash v_i : \delta_i(\tau)$, and
- if $\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}$, then $\vdash e_i : \delta_i(\tau)$.

We write $\delta(\Gamma)$ to mean a term substitution obtained from Γ by applying δ on the range of Γ , i.e.:

$$\text{dom}(\delta(\Gamma)) = \text{dom}(\Gamma) \text{ and } \forall x \in \text{dom}(\Gamma). \delta(\Gamma)(x) = \delta(\Gamma(x)).$$

Suppose that $\Delta, \Gamma \vdash e : \tau$, $\delta \models \Delta$, and $\gamma \models \delta(\Gamma)$. Then we write $\delta\gamma(e)$ to mean the application of γ and then δ to e . For example, suppose that $\delta(\alpha) = \mathbf{int}$, $\gamma(x) = n$ for some n , and $\alpha, x : \alpha \vdash \lambda y : \alpha. x : \alpha \rightarrow \alpha$, then $\delta\gamma(\lambda y : \alpha. x) = \lambda y : \mathbf{int}. n$. We write $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma \rrbracket_\rho$ for some $\rho \in \text{Rel}(\delta_1, \delta_2)$ when $\gamma_1 \models \delta_1(\Gamma)$, $\gamma_2 \models \delta_2(\Gamma)$, and $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \Gamma(x) \rrbracket_\rho$ for all $x \in \text{dom}(\Gamma)$.

$$\begin{array}{c}
\text{FR-INT} \frac{}{\langle n, n \rangle \in \llbracket \text{int} \rrbracket_\rho} \qquad \text{FR-PAIR} \frac{\langle v_1, v'_1 \rangle \in \llbracket \tau_1 \rrbracket_\rho \quad \langle v_2, v'_2 \rangle \in \llbracket \tau_2 \rrbracket_\rho}{\langle \langle v_1, v_2 \rangle, \langle v'_1, v'_2 \rangle \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho} \\
\\
\text{FR-FUN} \frac{\forall \langle v'_1, v'_2 \rangle \in \llbracket \tau_1 \rrbracket_\rho. \langle v_1 \ v'_1, v_2 \ v'_2 \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}}{\langle v_1, v_2 \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_\rho} \\
\\
\text{FR-VAR} \frac{\langle v_1, v_2 \rangle \in R \in \text{Rel}(\tau_1, \tau_2)}{\langle v_1, v_2 \rangle \in \llbracket \alpha \rrbracket_{\rho[\alpha \mapsto R]}} \\
\\
\text{FR-TERM} \frac{\vdash e_1 : \delta_1(\tau) \quad \vdash e_2 : \delta_2(\tau) \quad e_1 \rightarrow^* v_1 \quad e_2 \rightarrow^* v_2 \quad \langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_\rho}{\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}}
\end{array}$$

Fig. 2. The logical relation

Definition 1 (Logical equivalence). *Terms e and e' are logically equivalent at τ in Δ and Γ (written $\Delta, \Gamma \vdash e \sim e' : \tau$) if $\Delta, \Gamma \vdash e : \tau$, $\Delta, \Gamma \vdash e' : \tau$, and for all $\delta_1, \delta_2 \models \Delta$, all $\rho \in \text{Rel}(\delta_1, \delta_2)$, and all $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma \rrbracket_\rho$, we have $\langle \delta_1 \gamma_1(e), \delta_2 \gamma_2(e') \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}$.*

Theorem 1 (Abstraction [39]). *If $\Delta, \Gamma \vdash e : \tau$, then $\Delta, \Gamma \vdash e \sim e : \tau$.*

3 Declassification Policies

Confidentiality policies can be expressed by information flows of confidential sources to public sinks in programs. Confidential sources correspond to the secrets that the program receives and public sinks correspond to any results given to a public observer, a.k.a. the attacker. These flows can either be direct —e.g. if a function, whose result is public, receives a confidential value as input and directly returns the secret— or indirect —e.g. if a function, whose result is public, receives a confidential boolean value and returns 0 if the confidential value is false and 1 otherwise. Classification of program sources as confidential or public, a.k.a. *security policy*, must be given by the programmer or security engineer: for a given security policy the program is said to be secure for *noninterference* if public resources do not depend on confidential ones. Thus, noninterference for a program means total independence between public and confidential information. As simple and elegant as this information flow policy is, noninterference does not permit to consider as secure programs that purposely need to release information in a controlled way: for example a password-checker function that receives as confidential input a boolean value representing if the system password is equal to the user's input and returns 0 or 1 accordingly. In order to consider such intended dependences of public sinks from confidential sources, we need to consider more relaxed security policies than noninterference, a.k.a. *declassification policies*. Declassification security policies allow us to specify controlled ways to release confidential inputs [42].

Declassification policies that we consider in this work map confidential inputs to functions, namely *declassification functions*. These functions allow the programmer to specify what and how information can be released. The formal syntax for declassification functions in this work is given below,⁴ where n is an integer value, and \oplus represents primitive arithmetic operators.

| | |
|--|--------------------|
| $\tau ::= \mathbf{int} \mid \tau \rightarrow \tau$ | Types |
| $e ::= \lambda x : \tau. e \mid e \ e \mid x \mid n \mid e \oplus e$ | Terms |
| $f ::= \lambda x : \mathbf{int}. e$ | Declass. Functions |

The static and dynamic semantics are standard. To simplify the presentation we suppose that the applications of primitive operators on well-typed arguments terminates. Therefore, the evaluations of declassification functions on values terminate. A policy simply defines which are the confidential variables and their authorized declassifications. For policies we refrain from using concrete syntax and instead give a simple formalization that facilitates later definitions.

Definition 2 (Policy). *A policy \mathcal{P} is a tuple $\langle \mathbf{V}_{\mathcal{P}}, \mathbf{F}_{\mathcal{P}} \rangle$, where $\mathbf{V}_{\mathcal{P}}$ is a finite set of variables for confidential inputs, and $\mathbf{F}_{\mathcal{P}}$ is a partial mapping from variables in $\mathbf{V}_{\mathcal{P}}$ to declassification functions.*

For simplicity we require that if f appears in the policy then it is a closed term of type $\mathbf{int} \rightarrow \tau_f$ for some τ_f . In the definition of policies, if a confidential input is not associated with a declassification function, then it cannot be declassified.

Example 1 (Policy \mathcal{P}_{OE} using f). Consider policy \mathcal{P}_{OE} given by $\langle \mathbf{V}_{\mathcal{P}_{OE}}, \mathbf{F}_{\mathcal{P}_{OE}} \rangle$ where $\mathbf{V}_{\mathcal{P}_{OE}} = \{x\}$ and $\mathbf{F}_{\mathcal{P}_{OE}}(x) = f = \lambda x : \mathbf{int}. x \bmod 2$. Policy \mathcal{P}_{OE} states that only the parity of the confidential input x can be released to a public observer.

4 Type-based Declassification

In this section, we show how to encode declassification policies as standard types in the language of § 2, we define and we prove our free theorem. We consider a termination-sensitive [32] information flow security property,⁵ with declassification, called type-based relaxed noninterference (TRNI) and taken from Cruz et al [17]. It is important to notice that our development, in this section, studies the reuse for security of standard programming languages type systems together with soundness proofs for security for free by using the abstraction theorem. In contrast, Cruz et al [17] use a modified type system for security and prove soundness from scratch, without appealing to parametricity.

Through this section, we consider a fixed policy \mathcal{P} (see Def. 2) given by $\langle \mathbf{V}_{\mathcal{P}}, \mathbf{F}_{\mathcal{P}} \rangle$. We treat free variables in a program as inputs and, without loss of

⁴ In this paper, the type of confidential inputs is \mathbf{int} .

⁵ Our security property is termination sensitive but programs in the language always terminate. In the development for ML (in an appendix), programs may not terminate and the security property is also termination sensitive.

generality, we assume that there are two kinds of inputs: integer values, which are considered as confidential, and declassification functions, which are fixed according to policy. A public input can be encoded as a confidential input that can be declassified via the identity function. We consider terms without type variables as source programs. That is we consider terms e s.t. for all type substitutions δ , $\delta(e)$ is syntactically the same as e .⁶

4.1 Views and indistinguishability

We provide two term contexts to define TRNI, called the confidential view and public view. The first view represents an observer that can access confidential inputs, while the second one represents an observer that can only observe declassified inputs. The views are defined using fresh term and type variables.

Confidential view. Let $\mathbf{V}_\top = \{x \mid x \in \mathbf{V}_\mathcal{P} \setminus \text{dom}(\mathbf{F}_\mathcal{P})\}$ be the set of inputs that cannot be declassified. First we define the encoding for these inputs as a term context:

$$\Gamma_{C,\top}^\mathcal{P} \triangleq \{x : \mathbf{int} \mid x \in \mathbf{V}_\top\}.$$

Next, we specify the encoding of confidential inputs that can be declassified. To this end, define $\langle\langle _, _ \rangle\rangle_C$ as follows, where $f : \mathbf{int} \rightarrow \tau_f$ is in \mathcal{P} .

$$\langle\langle x, f \rangle\rangle_C \triangleq \{x : \mathbf{int}, x_f : \mathbf{int} \rightarrow \tau_f\}$$

Finally, we write $\Gamma_C^\mathcal{P}$ for the term context encoding the confidential view for \mathcal{P} .

$$\Gamma_C^\mathcal{P} \triangleq \Gamma_{C,\top}^\mathcal{P} \cup \bigcup_{x \in \text{dom}(\mathbf{F}_\mathcal{P})} \langle\langle x, \mathbf{F}_\mathcal{P}(x) \rangle\rangle_C.$$

We assume that, for any x , the variable x_f in the result of $\langle\langle x, \mathbf{F}_\mathcal{P}(x) \rangle\rangle_C$ is distinct from the variables in $\mathbf{V}_\mathcal{P}$, distinct from each other, and distinct from $x_{f'}$ for distinct f' . We make analogous assumptions in later definitions.

From the construction, $\Gamma_C^\mathcal{P}$ is a mapping, and for any $x \in \text{dom}(\Gamma_C^\mathcal{P})$, it follows that $\Gamma_C^\mathcal{P}(x)$ is a closed type. Therefore, $\Gamma_C^\mathcal{P}$ is well-formed for the empty set of type variables, so it can be used in typing judgments of the form $\Gamma_C^\mathcal{P} \vdash e : \tau$.

Example 2 (Confidential view). For \mathcal{P}_{OE} in Example 1, the confidential view is: $\Gamma_C^{\mathcal{P}_{OE}} = x : \mathbf{int}, x_f : \mathbf{int} \rightarrow \mathbf{int}$.

Public view. The basic idea is to encode policies by using type variables. First we define the encoding for confidential inputs that cannot be declassified. We define a set of type variables, $\Delta_{P,\top}^\mathcal{P}$ and a mapping $\Gamma_{P,\top}^\mathcal{P}$ for confidential inputs that cannot be declassified.

$$\Delta_{P,\top}^\mathcal{P} \triangleq \{\alpha_x \mid x \in \mathbf{V}_\top\} \quad \Gamma_{P,\top}^\mathcal{P} \triangleq \{x : \alpha_x \mid x \in \mathbf{V}_\top\}$$

⁶ An example of a term with type variables is $\lambda x : \alpha.x$. We can easily check that there exists a type substitutions δ s.t. $\delta(e)$ is syntactically different from e (e.g. for δ s.t. $\delta(\alpha) = \mathbf{int}$, $\delta(e) = \lambda x : \mathbf{int}.x$).

This gives the program access to x at an opaque type.

In order to define the encoding for confidential inputs that can be declassified, we define $\langle\langle -, - \rangle\rangle_P$:

$$\langle\langle x, f \rangle\rangle_P \triangleq \langle\{\alpha_f\}, \{x : \alpha_f, x_f : \alpha_f \rightarrow \tau_f\}\rangle$$

The first form will serve to give the program access to x only via function variable x_f that we will ensure is interpreted as the policy function f . We define a type context $\Delta_P^{\mathcal{P}}$ and term context $\Gamma_P^{\mathcal{P}}$ that comprise the public view, as follows.

$$\langle\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}}\rangle \triangleq \langle\Delta_{P,\top}^{\mathcal{P}}, \Gamma_{P,\top}^{\mathcal{P}}\rangle \cup \bigcup_{x \in \text{dom}(\mathbf{F}_{\mathcal{P}})} \langle\langle x, \mathbf{F}_{\mathcal{P}}(x) \rangle\rangle_P,$$

where $\langle S_1, S'_1 \rangle \cup \langle S_2, S'_2 \rangle = \langle S_1 \cup S_2, S'_1 \cup S'_2 \rangle$.

Example 3 (Public view). For \mathcal{P}_{OE} , the typing context in the public view has one type variable: $\Delta_P^{\mathcal{P}_{OE}} = \alpha_f$. The term context in the public view is $\Gamma_P^{\mathcal{P}_{OE}} = x : \alpha_f, x_f : \alpha_f \rightarrow \mathbf{int}$.

From the construction, $\Gamma_P^{\mathcal{P}}$ is a mapping, and for any $x \in \text{dom}(\Gamma_P^{\mathcal{P}})$, it follows that $\Gamma_P^{\mathcal{P}}(x)$ is well-formed in $\Delta_P^{\mathcal{P}}$ (i.e. $\Delta_P^{\mathcal{P}} \vdash \Gamma_P^{\mathcal{P}}(x)$). Thus, $\Gamma_P^{\mathcal{P}}$ is well-formed in the typing context $\Delta_P^{\mathcal{P}}$. Therefore, $\Delta_P^{\mathcal{P}}$ and $\Gamma_P^{\mathcal{P}}$ can be used in typing judgments of the form $\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}} \vdash e : \tau$.

Notice that in the public view of a policy, types of variables for confidential inputs are not **int**. Thus, the public view does not allow programs where concrete declassifiers are applied to confidential input variables even when the applications are semantically correct according to the policy (e.g. for \mathcal{P}_{OE} , the program $f\ x$ does not typecheck in the public view). Instead, programs should apply named declassifiers (e.g. for \mathcal{P}_{OE} , the program $x_f\ x$ is well-typed in the public view).

Indistinguishability. The security property TRNI is defined in a usual way, using partial equivalence relations called indistinguishability. To define indistinguishability, we define a type substitution $\delta_{\mathcal{P}}$ such that $\delta_{\mathcal{P}} \models \Delta_P^{\mathcal{P}}$, as follows:

$$\text{for all } \alpha_x, \alpha_f \text{ in } \Delta_P^{\mathcal{P}}, \text{ let } \delta_{\mathcal{P}}(\alpha_x) = \delta_{\mathcal{P}}(\alpha_f) = \mathbf{int}. \quad (1)$$

The inductive definition of indistinguishability for a policy \mathcal{P} is presented in Figure 3, where α_x and α_f are from $\Delta_P^{\mathcal{P}}$. Indistinguishability is defined for τ s.t. $\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}} \vdash \tau$. The definitions of indistinguishability for **int** and $\tau_1 \times \tau_2$ are straightforward. We say that two functions are indistinguishable at $\tau_1 \rightarrow \tau_2$ if on any indistinguishable inputs they generate indistinguishable outputs. Since we use α_x to encode confidential integer values that cannot be declassified, any integer values v_1 and v_2 are indistinguishable, according to rule Eq-Var1. Notice that $\delta_{\mathcal{P}}(\alpha_x) = \mathbf{int}$. Since we use α_f to encode confidential integer values that can be declassified via f where $\vdash f : \mathbf{int} \rightarrow \tau_f$, we say that $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_f]$ when $\langle f\ v_1, f\ v_2 \rangle \in \mathcal{I}_E[\tau_f]$.

$$\begin{array}{c}
\text{EQ-INT} \frac{}{\langle n, n \rangle \in \mathcal{I}_V[\mathbf{int}]} \quad \text{EQ-PAIR} \frac{\langle v_1, v'_1 \rangle \in \mathcal{I}_V[\tau_1] \quad \langle v_2, v'_2 \rangle \in \mathcal{I}_V[\tau_2]}{\langle \langle v_1, v_2 \rangle, \langle v'_1, v'_2 \rangle \rangle \in \mathcal{I}_V[\tau_1 \times \tau_2]} \\
\\
\text{EQ-FUN} \frac{\forall \langle v'_1, v'_2 \rangle : \langle v'_1, v'_2 \rangle \in \mathcal{I}_V[\tau_1]. \langle v_1 v'_1, v_2 v'_2 \rangle \in \mathcal{I}_E[\tau_2]}{\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \rightarrow \tau_2]} \\
\\
\text{EQ-VAR1} \frac{\vdash v_1, v_2 : \delta_{\mathcal{P}}(\alpha_x)}{\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_x]} \quad \text{EQ-VAR2} \frac{\vdash v_1, v_2 : \delta_{\mathcal{P}}(\alpha_f) \quad \langle f v_1, f v_2 \rangle \in \mathcal{I}_E[\tau_f]}{\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_f]} \\
\\
\text{EQ-TERM} \frac{\vdash e_1, e_2 : \delta_{\mathcal{P}}(\tau) \quad e_1 \rightarrow^* v_1 \quad e_2 \rightarrow^* v_2 \quad \langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau]}{\langle e_1, e_2 \rangle \in \mathcal{I}_E[\tau]}
\end{array}$$

Fig. 3. Indistinguishability

Example 4 (Indistinguishability). For \mathcal{P}_{OE} (of Example 1), two values v_1 and v_2 are indistinguishable at α_f when both of them are even numbers or odd numbers.

$$\mathcal{I}_V[\alpha_f] = \{ \langle v_1, v_2 \rangle \mid \vdash v_1 : \mathbf{int}, \vdash v_2 : \mathbf{int}, (v_1 \bmod 2) =_{\mathbf{int}} (v_2 \bmod 2) \}.$$

We write $e_1 =_{\mathbf{int}} e_2$ to mean that $e_1 \rightarrow^* v$ and $e_2 \rightarrow^* v$ for some integer value v .

Term substitutions γ_1 and γ_2 are called *indistinguishable w.r.t. \mathcal{P}* (denoted by $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$) if the following hold.

- $\gamma_1 \models \delta_{\mathcal{P}}(\Gamma_{\mathcal{P}}^{\mathcal{P}})$ and $\gamma_2 \models \delta_{\mathcal{P}}(\Gamma_{\mathcal{P}}^{\mathcal{P}})$,
- for all $x_f \in \text{dom}(\Gamma_{\mathcal{P}}^{\mathcal{P}})$, $\gamma_1(x_f) = \gamma_2(x_f) = f$,
- for all other $x \in \text{dom}(\Gamma_{\mathcal{P}}^{\mathcal{P}})$, $\langle \gamma_1(x), \gamma_2(x) \rangle \in \mathcal{I}_V[\Gamma_{\mathcal{P}}^{\mathcal{P}}(x)]$.

Note that each γ_i maps x_f to the specific function f in the policy. Input variables are mapped to indistinguishable values.

We now define type-based relaxed noninterference w.r.t. \mathcal{P} for a type τ well-formed in $\Delta_{\mathcal{P}}^{\mathcal{P}}$. It says that indistinguishable inputs lead to indistinguishable results.

Definition 3. A term e is *TRNI*(\mathcal{P}, τ) provided that $\Gamma_C^{\mathcal{P}} \vdash e$, and $\Delta_{\mathcal{P}}^{\mathcal{P}} \vdash \tau$, and for all $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$ we have $\langle \gamma_1(e), \gamma_2(e) \rangle \in \mathcal{I}_E[\tau]$.

Notice that if a term is well-typed in the public view then by replacing all type variables in it with **int**, we get a term which is also well-typed in the confidential view (that is, if $\Delta_{\mathcal{P}}^{\mathcal{P}}, \Gamma_{\mathcal{P}}^{\mathcal{P}} \vdash e : \tau$, then $\Gamma_C^{\mathcal{P}} \vdash \delta(e) : \delta(\tau)$ where δ maps all type variables in $\Delta_{\mathcal{P}}^{\mathcal{P}}$ to **int**). However, Definition 3 also requires that the term e is itself well-typed in the confidential view. This merely ensures that the definition is applied, as intended, to programs that do not contain type variables.

The definition of TRNI is indexed by a type for the result of the term. The type can be interpreted as constraining the observations to be made by the public observer. We are mainly interested in concrete output types, which express that

the observer can do whatever they like and has full knowledge of the result. Put differently, TRNI for an abstract type expresses security under the assumption that the observer is somehow forced to respect the abstraction. Consider the policy \mathcal{P}_{OE} (of Example 1) where x can be declassified via $f = \lambda x : \mathbf{int}. x \bmod 2$. As described in Example 3, $\Delta_{\mathcal{P}_{OE}}^{\mathcal{P}_{OE}} = \alpha_f$ and $\Gamma_{\mathcal{P}_{OE}}^{\mathcal{P}_{OE}} = x : \alpha_f, x_f : \alpha_f \rightarrow \mathbf{int}$. We have that the program x is $\text{TRNI}(\mathcal{P}_{OE}, \alpha_f)$ since the observer cannot do anything to x except for applying f to x which is allowed by the policy. This program, however, is not $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$ since the observer can apply any function of the type $\mathbf{int} \rightarrow \tau'$ (for some closed τ'), including the identity function, to x and hence can get the value of x .

Example 5. The program $x_f x$ is $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$. Indeed, for any arbitrary $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$, we have that $\gamma_1(x_f) = \gamma_2(x_f) = f = \lambda x : \mathbf{int}. x \bmod 2$, and $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_f]$, where $\gamma_1(x) = v_1$ and $\gamma_2(x) = v_2$ for some v_1 and v_2 . When we apply γ_1 and γ_2 to the program, we get respectively $v_1 \bmod 2$ and $v_2 \bmod 2$. Since $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_f]$, as described in Example 4, $(v_1 \bmod 2) =_{\mathbf{int}} (v_2 \bmod 2)$. Thus, $\langle \gamma_1(x_f x), \gamma_2(x_f x) \rangle \in \mathcal{I}_E[\mathbf{int}]$. Therefore, the program $x_f x$ satisfies the definition of TRNI.

4.2 Free theorem: typing in the public view implies security

In order to prove security “for free”, i.e., as consequence of Theorem 1, we define $\rho_{\mathcal{P}}$ as follows:

- for all $\alpha_x \in \Delta_{\mathcal{P}}^{\mathcal{P}}$, $\rho_{\mathcal{P}}(\alpha_x) = \mathcal{I}_V[\alpha_x]$,
- for all $\alpha_f \in \Delta_{\mathcal{P}}^{\mathcal{P}}$, $\rho_{\mathcal{P}}(\alpha_f) = \mathcal{I}_V[\alpha_f]$.

It is a relation on the type substitution $\delta_{\mathcal{P}}$ defined in Eqn. (1).

Lemma 2. $\rho_{\mathcal{P}} \in \text{Rel}(\delta_{\mathcal{P}}, \delta_{\mathcal{P}})$.

From Lemma 2, we can write $\llbracket \tau \rrbracket_{\rho_{\mathcal{P}}}$ or $\llbracket \tau \rrbracket_{\rho_{\mathcal{P}}}^{\text{ev}}$ for any τ such that $\Delta_{\mathcal{P}}^{\mathcal{P}} \vdash \tau$. We next establish the relation between $\llbracket \tau \rrbracket_{\rho}^{\text{ev}}$ and $\mathcal{I}_E[\tau]$: under the interpretation corresponding to the desired policy \mathcal{P} , they are equivalent. In other words, indistinguishability is an instantiation of the logical relation.

Lemma 3. For any τ such that $\Delta_{\mathcal{P}}^{\mathcal{P}} \vdash \tau$, we have $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_{\rho_{\mathcal{P}}}$ iff $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau]$, and also $\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_{\rho_{\mathcal{P}}}^{\text{ev}}$ iff $\langle e_1, e_2 \rangle \in \mathcal{I}_E[\tau]$.

By analyzing the type of $\Gamma_{\mathcal{P}}^{\mathcal{P}}(x)$, we can establish the relation of γ_1 and γ_2 when $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$.

Lemma 4. If $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$, then $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma_{\mathcal{P}}^{\mathcal{P}} \rrbracket_{\rho_{\mathcal{P}}}$.

The main result of this section is that a term is TRNI at τ if it has type τ in the public view that encodes the policy.

Theorem 2. If e has no type variables and $\Delta_{\mathcal{P}}^{\mathcal{P}}, \Gamma_{\mathcal{P}}^{\mathcal{P}} \vdash e : \tau$, then e is $\text{TRNI}(\mathcal{P}, \tau)$.

Proof. From the abstraction theorem (Theorem 1), for all $\delta_1, \delta_2 \models \Delta_P^{\mathcal{P}}$, for all $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket_{\rho}$, and for all $\rho \in \text{Rel}(\delta_1, \delta_2)$, it follows that

$$\langle \delta_1 \gamma_1(e), \delta_2 \gamma_2(e) \rangle \in \llbracket \tau \rrbracket_{\rho}^{\text{ev}}.$$

Consider $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$. Since $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\mathcal{P}]$, from Lemma 4, we have that $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket_{\rho_P}$. Thus, we have that $\langle \delta_P \gamma_1(e), \delta_P \gamma_2(e) \rangle \in \llbracket \tau \rrbracket_{\rho_P}^{\text{ev}}$. Since e has no type variable, we have that $\delta_P \gamma_i(e) = \gamma_i(e)$. Therefore, $\langle \gamma_1(e), \gamma_2(e) \rangle \in \llbracket \tau \rrbracket_{\rho_P}^{\text{ev}}$. Since $\langle \gamma_1(e), \gamma_2(e) \rangle \in \llbracket \tau \rrbracket_{\rho_P}^{\text{ev}}$, from Lemma 3, it follows that $\langle \gamma_1(e), \gamma_2(e) \rangle \in \mathcal{I}_E[\tau]$. In addition, since e has no type variable and $\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}} \vdash e : \tau$, we have that $\delta_P(\Gamma_P^{\mathcal{P}}) \vdash e : \delta_P(\tau)$ and hence, $\Gamma_C^{\mathcal{P}} \vdash e$. Therefore, e is TRNI(\mathcal{P}, τ).

Example 6 (Typing implies TRNI). Consider the policy \mathcal{P}_{OE} . As described in Examples 2 and 3, the confidential view $\Gamma_C^{\mathcal{P}_{OE}}$ is $x : \mathbf{int}, x_f : \mathbf{int} \rightarrow \mathbf{int}$ and the public view $\Delta_P^{\mathcal{P}_{OE}}, \Gamma_P^{\mathcal{P}_{OE}}$ is $\alpha_f, x : \alpha_f, x_f : \alpha_f \rightarrow \mathbf{int}$. We look at the program $x_f x$. We can easily verify that $\Gamma_C^{\mathcal{P}_{OE}} \vdash x_f x : \mathbf{int}$ and $\Delta_P^{\mathcal{P}_{OE}}, \Gamma_P^{\mathcal{P}_{OE}} \vdash x_f x : \mathbf{int}$. Therefore, by Theorem 2, the program is TRNI($\mathcal{P}_{OE}, \mathbf{int}$).

Example 7. If a program is well-typed in the confidential view but not TRNI(\mathcal{P}, τ) for some τ well-formed in the public view of \mathcal{P} , then the type of the program in the public view is not τ or the program is not well-typed in the public view. In policy \mathcal{P}_{OE} , from Example 6, the public view is $\alpha_f, x : \alpha_f, x_f : \alpha_f \rightarrow \mathbf{int}$. We first look at the program x that is not TRNI($\mathcal{P}_{OE}, \mathbf{int}$) since x itself is confidential and cannot be directly declassified. In the public view of the policy, the type of this program is α_f which is not \mathbf{int} . We now look at the program $x \text{ mod } 3$ that is not TRNI($\mathcal{P}_{OE}, \alpha_f$) since it takes indistinguishable inputs at α_f (e.g. 2 and 4) and produces results that are not indistinguishable at α_f (e.g. $2 = 2 \text{ mod } 3$, $1 = 4 \text{ mod } 3$, and $\langle 2, 1 \rangle \notin \mathcal{I}_V[\alpha_f]$). We can easily verify that this program is not well-typed in the public view since the type of x in the public view is α_f , while mod expects arguments of the \mathbf{int} type.

Remark 1 (Extension). Our encoding can be extended to support richer policies (details in appendix). To support policies where an input x can be declassified via two declassifiers $f : \mathbf{int} \rightarrow \tau_f$ and $g : \mathbf{int} \rightarrow \tau_g$ for some τ_f and τ_g , we use type variable $\alpha_{f,g}$ as the type for x and use $\alpha_{f,g} \rightarrow \tau_f$ and $\alpha_{f,g} \rightarrow \tau_g$ as types for x_f and x_g . To support policies where multiple inputs can be declassified via a declassifier, e.g. inputs x and y can be declassified via $f = \lambda z : \mathbf{int} \times \mathbf{int}. (\pi_1 z + \pi_2 z)/2$, we introduce a new term variable z which is corresponding to a tuple of two inputs x and y and we require that only z can be declassified. The type of z is α_f and two tuples $\langle v_1, v_2 \rangle$ and $\langle v'_1, v'_2 \rangle$ are indistinguishable at α_f when $f \langle v_1, v_2 \rangle = f \langle v'_1, v'_2 \rangle$.

5 Related Work

Typing secure information flow. Pottier and Simonet [35] implement Flow-Caml [36], the first type system for information flow analysis dealing with a

real-sized programming language (a large fragment of OCaml), and they prove soundness. In comparison with our results, we do not consider any imperative features; they do not consider any form of declassification, their type system significantly departs from ML typing, and their security proof is not based on an abstraction theorem. An interesting question is whether their type system can be translated to system F or some other calculus with an abstraction theorem. Flow-Caml provides type inference for security types. Our work relies on the Standard ML type system to enforce security. Standard ML provides type inference, which endows our approach with an inference mechanism. Barthe et al. [9] propose a modular method to reuse type systems and proofs for noninterference [43] for declassification. They also provide a method to conclude declassification soundness by using an existing noninterference theorem [40]. In contrast to our work, their type system significantly departs from standard typing rules, and does not make use of parametricity. Tse and Zdancewic [50] propose a security-typed language for robust declassification: declassification cannot be triggered unless there is a digital certificate to assert the proper authority. Their language inherits many features from System $F_{<}$ and uses monadic labels as in DCC [1]. In contrast to our work, security labels are based on the Decentralized Label Model (DLM) [30], and are not semantically unified with the standard safety types of the language. The Dependency Core Calculus (DCC) [1] expresses security policies using monadic types indexed on levels in a security lattice with the usual interpretation that flows are only allowed between levels in accordance with the ordering. DCC does not include declassification and the noninterference theorem of [1] is proved from scratch (not leveraging parametricity). While DCC is a theoretical calculus, its monadic types fit nicely with the monads and monad transformers used by the Haskell language for computational effects like state and I/O. Algehed and Russo [5] encode the typing judgment of DCC in Haskell using closed type families, one of the type system extensions supported by GHC that brings it close to dependent types. However, they do not prove security. Compared with type systems, relational logics can specify IF policy and prove more programs secure through semantic reasoning [31,8,24,10], but at the cost of more user guidance and less familiar notations. Aguirre et al [2] use relational higher order logic to prove soundness of DCC essentially by formalizing the semantics of DCC [1].

Connections between secure IF and type abstraction. Tse and Zdancewic [49] translate the recursion-free fragment of DCC to System F. The main theorem for this translation aims to show that parametricity of System F implies noninterference. Shikuma and Igarashi identify a mistake in the proof [44]; they also give a noninterference-preserving translation for a version of DCC to the simply-typed lambda calculus. Although they make direct use of a specific logical relation, their results are not obtained by instantiating a parametricity theorem. Bowman and Ahmed [12] finally provide a translation from the recursion-free fragment of DCC to System F_{ω} , proving that parametricity implies noninterference, via a correctness theorem for the translation (which is akin to a full abstraction property). Bowman and Ahmed’s translation makes essential use of

the power of System F_ω to encode judgments of DCC. Algehed and Bernardy [4] translate a label-polymorphic variant DCC (without recursion) into the calculus of constructions (CC) and prove noninterference directly from a parametricity result for CC [11]. The authors note that it is not obvious this can be extended to languages with nontermination or other effects. Their results have been checked in Agda and the presentation achieves elegance owing to the fact that parametricity and noninterference can be explicitly defined in dependent type theory; indeed, CC terms can represent proof of parametricity [11]. Our goals do not necessitate a system like DCC for policy, raising the question of whether a simpler target type system can suffice for security policies expressed differently from DCC. We answer the question in the affirmative, and believe our results for polymorphic lambda (and for ML) provide transparent explication of noninterference by reduction to parametricity. The preceding works on DCC are “translating noninterference to parametricity” in the sense of translating both programs and types. The implication is that one might leverage an existing type checker by translating both a program and its security policy into another program such that its typability implies the original conforms to policy. Our work aims to cater more directly for practical application, by minimizing the need to translate the program and hence avoiding the need to prove the correctness of a translation. Cruz et al. [17] show that type abstraction implies relaxed noninterference. Similar to ours, their definition of relaxed noninterference is a standard extensional semantics, using partial equivalence relations. This is in contrast with Li and Zdancewic [27] where the semantics is entangled with typability.

Protzenko et al. [37] propose to use abstract types as the types for secrets and use standard type systems for security. This is very close in spirit to our work. Their soundness theorem is about a property called “secret independence”, very close to noninterference. In contrast to our work, there is no declassification and no use of the abstraction theorem. Rajani and Garg [38] connect fine- and coarse-grained type systems for information flow in a lambda calculus with general references, defining noninterference (without declassification) as a step-indexed Kripke logical relation that expresses indistinguishability. Further afield, a connection between security and parametricity is made by Devriese et al [18], featuring a negative result: System F cannot be compiled to the the Sumii-Pierce calculus of dynamic sealing [47] (an idealized model of a cryptographic mechanism). Finally, information flow analyses have also been put at the service of parametricity [54].

Abstraction theorems for other languages. Parametricity remains an active area of study [46]. Vytiniotis and Weirich [52] prove the abstraction theorem for R_ω , which extends F_ω with constructs that are useful for programming with type equivalence propositions. Rossberg et al [41] show another path to parametricity for ML modules, by translating them to F_ω . Crary’s result [14] covers a large fragment of ML but without references and mutable state. Abstraction theorems have been given for mutable state, based on ownership types [6] and on more semantically based reasoning [3,20,7,48].

6 Discussion and Conclusion

In this work, we show how to express declassification policies by using standard types of the simply typed lambda calculus. By means of parametricity, we prove that type checking implies relaxed noninterference, showing a direct connection between declassification and parametricity. Our approach should be applicable to other languages that have an abstraction theorem (e.g. [7,3,20,48]) with the potential benefit of strong security assurance from off-the-shelf type checkers. In particular, we demonstrate (in an appendix) that the results can be extended to a large fragment of ML including general recursion. Although in this paper we demonstrate our results using confidentiality and declassification, our approach applies as well to integrity and endorsement, as they have been shown to be information flow properties analog to confidentiality [26,23,21,22].

The simple encodings in the preceding sections do not support computation and output at multiple levels. For example, consider a policy where x is a confidential input that can be declassified via f and we also want to do the computation $x + 1$ of which the result is at confidential level. Clearly, $x + 1$ is ill-typed in the public interface. We provide (in an appendix) more involved encodings supporting computation at multiple levels. To have an encoding that support multiple levels, we add universally quantified types $\forall\alpha.\tau$ to the language presented in §2. However, this goes against our goal of minimizing complexity of translation. Observe that many applications are composed of programs which, individually, do not output at multiple levels; for example, the password checker, and data mining computations using sensitive inputs to calculate aggregate or statistical information. For these the simpler encoding suffices.

Vanhoef et al. [51] and others have proposed more expressive declassification policies than the ones in Li and Zdancewic [27]: policies that keep state and can be written as programs. We speculate that TRNI for stateful declassification policies can be obtained for free in a language with state—indeed, our work provides motivation for development of abstraction theorems for such languages.

Acknowledgements. We thank anonymous reviewers for their suggestions. This work was partially supported by CISC ANR-17-CE25-0014-01, IPL SPAI, the European Union’s Horizon 2020 research and innovation programme under grant agreement No 830892, and US NSF award CNS 1718713.

References

1. Abadi, M., Banerjee, A., Heintze, N., Riecke, J.G.: A core calculus of dependency. In: ACM POPL. pp. 147–160 (1999)
2. Aguirre, A., Barthe, G., Gaboardi, M., Garg, D., Strub, P.: A relational logic for higher-order programs. PACMPL **1**(ICFP), 21:1–21:29 (2017)
3. Ahmed, A., Dreyer, D., Rossberg, A.: State-dependent representation independence. In: ACM POPL. pp. 340–353 (2009)
4. Alghed, M., Bernardy, J.: Simple noninterference from parametricity. PACMPL **3**(ICFP), 89:1–89:22 (2019)

5. Alghed, M., Russo, A.: Encoding DCC in Haskell. In: Workshop on Programming Languages and Analysis for Security. pp. 77–89 (2017)
6. Banerjee, A., Naumann, D.A.: Ownership confinement ensures representation independence for object-oriented programs. *Journal of the ACM* **52**(6), 894–960 (2005)
7. Banerjee, A., Naumann, D.A.: State based encapsulation for modular reasoning about behavior-preserving refactorings. In: *Aliasing in Object-oriented Programming*. Springer State-of-the-art Surveys (2012)
8. Banerjee, A., Naumann, D.A., Nikouei, M.: Relational logic with framing and hypotheses. In: *FSTTCS. LIPIcs*, vol. 65, pp. 11:1–11:16 (2016)
9. Barthe, G., Cavadini, S., Rezk, T.: Tractable enforcement of declassification policies. In: *IEEE Computer Security Foundations Symposium*. pp. 83–97 (2008)
10. Beckert, B., Ulbrich, M.: Trends in relational program verification. In: Müller, P., Schaefer, I. (eds.) *Principled Software Development - Essays Dedicated to Arnd Poetzsch-Heffter on the Occasion of his 60th Birthday*. pp. 41–58. Springer (2018)
11. Bernardy, J.P., Jansso, P., Paterson, R.: Proofs for free: Parametricity for dependent types. *Journal of Functional Programming* **22**(2), 107–152 (2012)
12. Bowman, W.J., Ahmed, A.: Noninterference for free. In: *ICFP*. pp. 101–113 (2015)
13. Crary, K.: Logical relations and a case study in equivalence checking. In: Pierce, B.C. (ed.) *Advanced Topics in Types and Programming Languages*, chap. 6, pp. 245–289. The MIT Press (2005)
14. Crary, K.: Modules, abstraction, and parametric polymorphism. In: *ACM POPL*. pp. 100–113 (2017)
15. Crary, K.: Modules, abstraction, and parametric polymorphism - coq development for popl 2017. <https://www.cs.cmu.edu/~crary/papers/2016/mapp.tgz> (2017)
16. Crary, K.: Fully abstract module compilation. In: *ACM POPL*. vol. 3, pp. 10:1–10:29 (2019)
17. Cruz, R., Rezk, T., Serpette, B.P., Tanter, É.: Type abstraction for relaxed non-interference. In: *ECOOP*. pp. 7:1–7:27 (2017)
18. Devriese, D., Patrignani, M., Piessens, F.: Parametricity versus the universal type. *PACMPL* **2**(POPL), 38:1–38:23 (2018)
19. Dreyer, D.: Understanding and Evolving the ML Module System. Ph.D. thesis, Carnegie Mellon University (2005)
20. Dreyer, D., Neis, G., Rossberg, A., Birkedal, L.: A relational modal logic for higher-order stateful ADTs. In: *ACM POPL*. pp. 185–198 (2010)
21. Fournet, C., Guernic, G.L., Rezk, T.: A security-preserving compiler for distributed programs: from information-flow policies to cryptographic mechanisms. In: *ACM Conference on Computer and Communications Security, CCS* (2009)
22. Fournet, C., Planul, J., Rezk, T.: Information-flow types for homomorphic encryptions. In: *ACM CCS*. pp. 351–360 (2011)
23. Fournet, C., Rezk, T.: Cryptographically sound implementations for typed information-flow security. In: *ACM POPL* (2008)
24. Grimm, N., Maillard, K., Fournet, C., Hritcu, C., Maffei, M., Protzenko, J., Ramananandro, T., Rastogi, A., Swamy, N., Béguelin, S.Z.: A monadic framework for relational verification: applied to information security, program equivalence, and optimizations. In: *Certified Programs and Proofs*. pp. 130–145 (2018)
25. Harper, R.: *Practical foundations for programming languages*. Cambridge University Press (2016)
26. Li, P., Mao, Y., Zdancewic, S.: Information integrity policies. In: *In Proceedings of the Workshop on Formal Aspects in Security and Trust (FAST)* (2003)
27. Li, P., Zdancewic, S.: Downgrading policies and relaxed noninterference. In: *ACM POPL*. pp. 158–170 (2005)

28. Mitchell, J.C.: *Foundations for Programming Languages*. MIT Press (1996)
29. Myers, A.C.: Jif homepage. <http://www.cs.cornell.edu/jif/> (accessed July 2018)
30. Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. *ACM Trans. on Software Engineering and Methodology* **9**, 410–442 (Oct 2000)
31. Nanevski, A., Banerjee, A., Garg, D.: Dependent type theory for verification of information flow and access control policies. *ACM Trans. Program. Lang. Syst.* **35**(2), 6 (2013)
32. Ngo, M., Piessens, F., Rezk, T.: Impossibility of precise and sound termination-sensitive security enforcements. In: 2018 IEEE Symposium on Security and Privacy, SP. IEEE Computer Society (2018)
33. Petricek, T.: What we talk about when we talk about monads. *Programming Journal* **2**, 12 (2018)
34. Pitts, A.M.: Typed operational reasoning. In: Pierce, B.C. (ed.) *Advanced Topics in Types and Programming Languages*, chap. 7, pp. 245–289. The MIT Press (2005)
35. Pottier, F., Simonet, V.: Information flow inference for ML. In: *ACM POPL*. pp. 319–330 (2002)
36. Pottier, F., Simonet, V.: Flowcaml homepage. <https://www.normalesup.org/~simonet/soft/flowcaml/index.html> (accessed July 2018)
37. Protzenko, J., Zinzindohoué, J.K., Rastogi, A., Ramananandro, T., Wang, P., Béguelin, S.Z., Delignat-Lavaud, A., Hritcu, C., Bhargavan, K., Fournet, C., Swamy, N.: Verified low-level programming embedded in F. *PACMPL* **1**(ICFP), 17:1–17:29 (2017)
38. Rajani, V., Garg, D.: Types for information flow control: Labeling granularity and semantic models. In: *IEEE Computer Security Foundations Symposium* (2018)
39. Reynolds, J.C.: Types, abstraction and parametric polymorphism. In: *IFIP Congress*. pp. 513–523 (1983)
40. Rezk, T.: *Verification of confidentiality policies for mobile code*. Ph.D. thesis, University of Nice-Sophia Antipolis (2006)
41. Rossberg, A., Russo, C.V., Dreyer, D.: F-ing modules. *J. Funct. Program.* **24**(5), 529–607 (2014)
42. Sabelfeld, A., Sands, D.: Declassification: Dimensions and principles. *Journal of Computer Security* **17**(5), 517–548 (2009)
43. Santos, J.F., Jensen, T.P., Rezk, T., Schmitt, A.: Hybrid typing of secure information flow in a javascript-like language. In: *Trustworthy Global Computing - 10th International Symposium, TGC* (2015)
44. Shikuma, N., Igarashi, A.: Proving noninterference by a fully complete translation to the simply typed lambda-calculus. *Logical Methods in Comp. Sci.* **4**(3) (2008)
45. Standard ML of New Jersey homepage. <https://www.smlnj.org/>
46. Sojakova, K., Johann, P.: A general framework for relational parametricity. In: *IEEE Symp. on Logic in Computer Science*. pp. 869–878 (2018)
47. Sumii, E., Pierce, B.C.: A bisimulation for dynamic sealing. In: *ACM POPL*. pp. 161–172 (2004)
48. Timany, A., Stefanescu, L., Krogh-Jespersen, M., Birkedal, L.: A logical relation for monadic encapsulation of state: Proving contextual equivalences in the presence of runST. *Proc. ACM Program. Lang.* **2**(POPL), 64:1–64:28 (Dec 2017)
49. Tse, S., Zdancewic, S.: Translating dependency into parametricity. In: *International Conference on Functional Programming*. pp. 115–125 (2004)
50. Tse, S., Zdancewic, S.: A design for a security-typed language with certificate-based declassification. In: *ESOP*. pp. 279–294 (2005)

- 51. Vanhoef, M., Groef, W.D., Devriese, D., Piessens, F., Rezk, T.: Stateful declassification policies for event-driven programs. In: IEEE Computer Security Foundations Symposium. pp. 293–307 (2014)
- 52. Vytiniotis, D., Weirich, S.: Parametricity, type equality, and higher-order polymorphism. *J. Funct. Program.* **20**(2), 175–210 (2010)
- 53. Wadler, P.: Theorems for free! In: International Conference on Functional Programming. pp. 347–359 (1989)
- 54. Washburn, G., Weirich, S.: Generalizing parametricity using information-flow. In: IEEE Symp. on Logic in Computer Science. pp. 62–71 (2005)

Contents of appendix

§A describes extensions for more expressive policies, in terms of the encoding in §4. §B is an overview of our results for the ML module calculus and §C is an overview of our encoding for computation at multiple security levels. §D provides proofs for §2 and §E provides proofs for §4. The following sections present the ML encoding (§F–§H) and the multi-level encoding (§I–§J) in detail.

A Extensions

The extensions in this section are corresponding to the encoding in §4.

A.1 Declassification policies

Variations of our encoding can support richer declassification policies and accept more secure programs. We consider two ways to extend our encoding.

More declassification functions. The notation in [27] labels an input with a set of declassification functions, so in general an input can be declassified in more than one way. To show how this can be accomodated, we present an extension for a policy \mathcal{P} where $\mathbf{V}_{\mathcal{P}} = \{x\}$, and x can be declassified via f or g for some f and g , where $\vdash f : \mathbf{int} \rightarrow \tau_f$ and $\vdash g : \mathbf{int} \rightarrow \tau_g$. The confidential view and the public view for this policy are as below:

$$\begin{aligned} \Gamma_C^{\mathcal{P}} &= x : \mathbf{int}, x_f : \mathbf{int} \rightarrow \tau_f, x_g : \mathbf{int} \rightarrow \tau_g \\ \Delta_P^{\mathcal{P}} &= \alpha_{f,g} \\ \Gamma_P^{\mathcal{P}} &= x : \alpha_{f,g}, x_f : \alpha_{f,g} \rightarrow \tau_f, x_g : \alpha_{f,g} \rightarrow \tau_g \end{aligned}$$

We now have a new definition of indistinguishability. The definition is similar to the one presented in §4, except that we add a new rule for $\alpha_{f,g}$.

$$\text{EQ-VAR4} \frac{\vdash v_1, v_2 : \mathbf{int} \quad \langle f \ v_1, f \ v_2 \rangle \in \mathcal{I}_E[\tau_f] \quad \langle g \ v_1, g \ v_2 \rangle \in \mathcal{I}_E[\tau_g]}{\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_{f,g}]}$$

With the new encoding and the new definition of indistinguishability, we can define $\text{TRNI}(\mathcal{P}, \tau)$ as in Definition 3. From the abstraction theorem, we again obtain that for any program e , if $\Gamma_C^{\mathcal{P}} \vdash e$, and $\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}} \vdash e : \tau$, then e is $\text{TRNI}(\mathcal{P}, \tau)$.

For example, we consider programs $e_1 = x_f \ x$ and $e_2 = x_g \ x$. These two programs are well-typed in both views of \mathcal{P} , and in the public view, their types are respectively τ_f and τ_g . Thus, e_1 is $\text{TRNI}(\mathcal{P}, \tau_f)$, and e_2 is $\text{TRNI}(\mathcal{P}, \tau_g)$.

Using an equivalent function to declassify. In most type systems for declassification, the declassifier function or expression must be identical to the one in the policy. Indeed, policy is typically expressed by writing a “declassify” annotation on the expression [42]. However, the type system presented in [27, § 5] is more permissive: it accepts a declassification if it is semantically equivalent to the policy function, according to a given syntactically defined approximation of equivalence. Verification tools can go even further in reasoning with semantic equivalence [31,24], but any automated checker is limited due to undecidability of semantic equivalence.

We consider a policy \mathcal{P} where there are two confidential inputs x and y , x can be declassified via f , and y can be declassified via g , $f : \mathbf{int} \rightarrow \tau$, and $g : \mathbf{int} \rightarrow \tau$ for some τ . Suppose that there exists a function a s.t. $f \circ a = g$ semantically. With the encoding in §4, we accept $g \ y$, or rather $x_g \ y$, but we cannot accept $f(a \ y)$ even though it is semantically the same.

To accept programs like $f(a \ y)$, based on the idea of the first extension, we encode the policy as below, where y is viewed as a confidential input that can be declassified via g or $f \circ a$. Note that in the following encoding, we have two type variables: $\alpha_{g, f \circ a}$ for the confidential input, and α_f for the result of $x_a \ x$ which can be declassified via f .

$$\begin{aligned} \Gamma_C^{\mathcal{P}} &= x : \mathbf{int}, x_f : \mathbf{int} \rightarrow \tau, y : \mathbf{int}, y_g : \mathbf{int} \rightarrow \tau, y_a : \mathbf{int} \rightarrow \mathbf{int} \\ \Delta_P^{\mathcal{P}} &= \alpha_f, \alpha_{g, f \circ a} \\ \Gamma_P^{\mathcal{P}} &= x : \alpha_f, x_f : \alpha_f \rightarrow \tau, y : \alpha_{g, f \circ a}, y_g : \alpha_{g, f \circ a} \rightarrow \tau, x_a : \alpha_{g, f \circ a} \rightarrow \alpha_f \end{aligned}$$

Indistinguishability for this policy is defined similarly to the one in Section 4, except that we have the following rule for $\alpha_{g, f \circ a}$.

$$\text{EQ-VAR6} \frac{\vdash v_1 : \mathbf{int} \quad \vdash v_2 : \mathbf{int} \quad \langle g \ v_1, g \ v_2 \rangle \in \mathcal{I}_E[\tau]}{\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_{g, f \circ a}]}$$

As in the first extension, we can define TRNI for a type τ well-formed in $\Delta_P^{\mathcal{P}}$ and we have the free theorem stating that if $\Gamma_C^{\mathcal{P}} \vdash e$, and $\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}} \vdash e : \tau$, then e is TRNI(\mathcal{P}, τ).

W.r.t. the new encoding, both $y_g \ y$ and $x_f(x_a \ y)$ are well-typed in the public view. In other words, we accepts both $y_g \ y$ and $x_f(x_a \ y)$.

Notice that as discussed in [27], the problem of establishing relations between declassification functions in general is undecidable. Thus, the relations should be provided or can be found in a predefined amount of time. Otherwise, the relations are not used in the encoding and programs like $x_f(y_a \ y)$ will not typecheck.

A.2 Global policies

The policies considered in §4 and §A.1 are corresponding to local policies in [27]. We now consider policies where a declassifier can involve more than one

confidential input. To be consistent with [27], we call such policies global policies. For simplicity, in this subsection, we consider a policy \mathcal{P} where there are two confidential inputs, x_1 and x_2 , which can be declassified via f of the type $\mathbf{int}_1 \times \mathbf{int}_2 \rightarrow \tau_f$.⁷ Notice that here we use subscripts for the input type of f to mean that the confidential input x_i is corresponding to i -th element of an input of f .

Example 8 (Average can be declassified). We consider the policy \mathcal{P}_{Ave} where there are two confidential inputs x_1 and x_2 and their average can be declassified. That is x_1 and x_2 can be declassified via $f = \lambda x : \mathbf{int} \times \mathbf{int}.(\pi_1 x + \pi_2 x)/2$.

In our encoding, we need to maintain the correspondence between inputs and arguments of the declassifier since we want to prevent laundering attacks [42]. A laundering attack occurs, for example, when the declassifier f is applied to $\langle x_1, x_1 \rangle$, since then the value of x_1 is leaked.

In the general case, to encode the requirement that a specific n -tuple of confidential inputs can be declassified via f , we introduce a new variable y . The basic idea is that y is corresponding to that n -tuple of confidential inputs, x_i cannot be declassified, and only y can be declassified via f . Therefore, the confidential and public views are as below, where for readability we show the case $n = 2$.

$$\begin{aligned} \Gamma_C^{\mathcal{P}} &\triangleq \{x_1 : \mathbf{int}, x_2 : \mathbf{int}, y : \mathbf{int} \times \mathbf{int}, y_f : \mathbf{int} \times \mathbf{int} \rightarrow \tau_f\} \\ \Delta_P^{\mathcal{P}} &\triangleq \{\alpha_{x_1}, \alpha_{x_2}, \alpha_f\} \\ \Gamma_P^{\mathcal{P}} &\triangleq \{x_1 : \alpha_{x_1}, x_2 : \alpha_{x_2}, y : \alpha_f, y_f : \alpha_f \rightarrow \tau_f\} \end{aligned}$$

For each $i \in \{1, \dots, n\}$, since x_i cannot be declassified, the indistinguishability for α_{x_i} is the same as the one for α_x described in Fig. 3. Since y corresponds to the tuple of confidential inputs and only it can be declassified via f , indistinguishability for the type of y in the public view α_f is as below (again, case $n = 2$).

$$\text{EQ-VAR5} \frac{\vdash v, v' : \mathbf{int} \times \mathbf{int} \quad \langle f \ v, f \ v' \rangle \in \mathcal{I}_E[\tau_f]}{\langle v, v' \rangle \in \mathcal{I}_V[\alpha_f]}$$

We next encode the correspondence between inputs and argument of the declassifier. We say that a term substitution γ is *consistent* w.r.t. $\Gamma_P^{\mathcal{P}}$ if $\gamma \models \delta_{\mathcal{P}}(\Gamma_P^{\mathcal{P}})$ and in addition, for all $i \in \{1, 2\}$, $\pi_i(\gamma(y)) = \gamma(x_i)$. As we can see, the additional condition takes care of the correspondence of inputs and the arguments of the intended declassifier.

We next define the type substitution and indistinguishable term substitutions for \mathcal{P} . We say that $\delta_{\mathcal{P}} \models \Delta_P^{\mathcal{P}}$ when $\delta_{\mathcal{P}}(\alpha_f) = \mathbf{int} \times \mathbf{int}$ and for all α_{x_i} , $\delta_{\mathcal{P}}(\alpha_{x_i}) = \mathbf{int}$. We say that two term substitutions γ_1 and γ_2 are indistinguishable w.r.t.

⁷ We can extend the encoding presented in this section to have policies where different subsets of $\mathbf{V}_{\mathcal{P}}$ can be declassified and to have more than one declassifier associated with a set of confidential inputs.

\mathcal{P} (denoted by $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V[\llbracket \mathcal{P} \rrbracket]$) if γ_1 and γ_2 are consistent w.r.t. $\Gamma_P^{\mathcal{P}}$, $\gamma_1(y_f) = \gamma_2(y_f) = f$, for all other $x \in \text{dom}(\Gamma_P^{\mathcal{P}})$, $\langle \gamma_1(x), \gamma_2(x) \rangle \in \mathcal{I}_V[\llbracket \Gamma_P^{\mathcal{P}}(x) \rrbracket]$.

Then we can define $\text{TRNI}(\mathcal{P}, \tau)$ as in Def. 3 (except that we use the new definition of indistinguishable term substitutions). We also have the free theorem stating that if e has no type variable and $\Delta_P^{\mathcal{P}}, \Gamma_P^{\mathcal{P}} \vdash e : \tau$, then e is $\text{TRNI}(\mathcal{P}, \tau)$. The proof goes through without changes.

Example 9 (Average can be declassified - cont.). Here we present the encoding for the policy \mathcal{P}_{Ave} described in Example 8. The confidential and public views for this policy is as below:

$$\begin{aligned} \Gamma_C^{\mathcal{P}_{\text{Ave}}} &\triangleq \{x_1 : \mathbf{int}, x_2 : \mathbf{int}, y : \mathbf{int} \times \mathbf{int}, y_f : \mathbf{int} \times \mathbf{int} \rightarrow \mathbf{int}\} \\ \Delta_P^{\mathcal{P}_{\text{Ave}}} &\triangleq \{\alpha_{x_1}, \alpha_{x_2}, \alpha_f\} \\ \Gamma_P^{\mathcal{P}_{\text{Ave}}} &\triangleq \{x_1 : \alpha_{x_1}, x_2 : \alpha_{x_2}, y : \alpha_f, y_f : \alpha_f \rightarrow \mathbf{int}\} \end{aligned}$$

We can easily check that the program $y_f y$ is $\text{TRNI}(\mathcal{P}_{\text{Ave}}, \mathbf{int})$; it is well-typed in both views, and in the public view its type is \mathbf{int} .

B TRNI for Module Calculus: summary

This section recapitulates the development of §4 but using an encoding suited to the module calculus of Crary and Dreyer [19,14].⁸ It is a core calculus that models Standard ML including higher order generative and applicative functors, sharing constraints (via singleton kinds), and sealing. Sealing ascribes a signature to a module expression and thereby enforces data abstraction.

The syntax is in Fig. 4. The calculus has static expressions: kinds (k), constructors (c) and signatures (σ), and dynamic expressions: terms (e) and modules (M). The full formal system is given in §F; here we sketch highlights.

The unit kind 1 has only the unit constructor \star . The base kind, \mathbf{T} , is for types that can be used to classify terms. By convention, we use the metavariable τ for constructors that are types (i.e. of the kind \mathbf{T}). The singleton kind $\mathbf{S}(c)$ classifies constructors that are definitionally equivalent to c . In addition, we have higher kinds: dependent functions $\Pi \alpha : k_1.k_2$ and dependent pairs $\Sigma \alpha : k_1.k_2$.

The syntax for terms is standard and includes general recursion ($\text{fix}_\tau e$). Module expressions include unit module (\star), pairing/projection, atomic modules with a single static or dynamic component ($\langle \langle c \rangle \rangle, \langle \langle e \rangle \rangle$), generative and applicative functors ($\lambda^{\text{gn}} \alpha / m : \sigma.M, \lambda^{\text{ap}} \alpha / m : \sigma.M$, the applications of which are written resp. $M_1 M_2, M_1 \cdot M_2$), and unpacking ($\text{unpack}[\alpha, x] = e \text{ in } (M : \sigma)$). While term binding is as usual ($\text{let } x = e \text{ in } M$), the module binding construct is unusual: $\text{let } \alpha / m = M_1 \text{ in } (M_2 : \sigma)$ binds a pair of names, where constructor variable α is used to refer to the static part of M_1 (and m to the full module). This is used to handle the phase distinction between compile-time and run-time expressions.

⁸ Our only change is to add \mathbf{int} and arithmetic primitives, for examples.

| $k ::=$ | kind | $e ::=$ | term |
|---|------------------------|---|---------------------|
| 1 | unit kind | x | term variable |
| \mathbf{T} | base kind | \star | unit term |
| $\mathbf{S}(c)$ | singleton kind | n | integer literal |
| $\Pi\alpha : k.k$ | dependent functions | $\lambda x : \tau.e \mid e e$ | lambda, application |
| $\Sigma\alpha : k.k$ | dependent pairs | $\langle e, e \rangle$ | pair |
| $c, \tau ::=$ | type constructor | $\pi_1 e \mid \pi_2 e$ | projection |
| α | constructor variable | $\Lambda\alpha : k.e$ | polymorphic fun. |
| \star | unit constructor | $e[c]$ | polymorphic app. |
| $\lambda\alpha : k.c \mid c c$ | lambda, application | $\mathbf{pack}[c, e] \text{ as } \exists\alpha : k.\tau$ | existential package |
| $\langle c, c \rangle$ | pair | $\mathbf{unpack}[\alpha, x] = e \text{ in } e$ | unpack |
| $\pi_1 c \mid \pi_2 c$ | projection | $\mathbf{fix}_\tau e$ | recursion |
| \mathbf{unit} | unit type | $\mathbf{let } x = e \text{ in } e$ | term binding |
| \mathbf{int} | int type | $\mathbf{let } \alpha/m = M \text{ in } e$ | module binding |
| $\tau_1 \rightarrow \tau_2$ | functions | $\mathbf{Ext } M$ | extraction |
| $\tau_1 \times \tau_2$ | products | $M ::=$ | module |
| $\forall\alpha : k.\tau$ | universal | m | module variable |
| $\exists\alpha : k.\tau$ | existential | \star | unit module |
| $\sigma ::=$ | signature | $\langle c \rangle$ | atomic module |
| 1 | unit signature | $\langle e \rangle$ | atomic module |
| $\langle k \rangle$ | atomic signature | $\lambda^{\text{gn}}\alpha/m : \sigma.M$ | generative functor |
| $\langle \tau \rangle$ | atomic signature | $M M$ | generative app. |
| $\Pi^{\text{gn}}\alpha : \sigma.\sigma$ | generative functors | $\lambda^{\text{ap}}\alpha/m : \sigma.M$ | applicative functor |
| $\Pi^{\text{ap}}\alpha : \sigma.\sigma$ | applicative functors | $M \cdot M$ | applicative app. |
| $\Sigma\alpha : \sigma.\sigma$ | pairs | $\langle M, M \rangle$ | pair |
| $\Gamma ::=$ | context | $\pi_1 M \mid \pi_2 M$ | projection |
| . | empty context | $\mathbf{unpack}[\alpha, x] = e \text{ in } (M : \sigma)$ | unpack |
| $\Gamma, \alpha : k$ | constructor hypothesis | $\mathbf{let } x = e \text{ in } M$ | term binding |
| $\Gamma, x : \tau$ | term hypothesis | $\mathbf{let } \alpha/m = M \text{ in } (M : \sigma)$ | module binding |
| $\Gamma, \alpha/m : \sigma$ | module hypothesis | $M :> \sigma$ | sealing |

Fig. 4. Module calculus

A signature describes an interface for a module. Signatures include unit signature, atomic kind and atomic type signature, generative and applicative functors, and dependent pairs ($\Sigma\alpha : \sigma_1.\sigma_2$). A signature σ is *transparent* when it exposes the implementation of the static part of modules of σ . A signature σ is *opaque* when it hides some information about the static part of modules of σ .

The sealing construct, $M \text{ :> } \sigma$, ascribes a signature to the module in the sense of enforcing σ as an abstraction boundary.

Abstraction theorem. The static semantics includes judgments $\vdash \Gamma \text{ ok}$, $\Gamma \vdash e : \tau$, $\Gamma \vdash_P M : \sigma$, and $\Gamma \vdash_I M : \sigma$ for resp. well-formed context, well-typed term, pure well-formed module, and impure well-formed module. The pure and impure judgment forms roughly correspond to unsealed and sealed modules; the formal system treats sealing as an effect, introduced by application of a generative functor as well as by the sealing construct.

The dynamic semantics is call-by value, with these values:

$$\begin{array}{ll}
 v := x \mid \star \mid n \mid \lambda x : \tau. e \mid \langle v, v \rangle \mid \Lambda \alpha : k. e & \text{Term values} \\
 \mid \text{pack}[c, v] \text{ as } \exists \alpha : k. \tau & \\
 V := m \mid \star \mid \langle c \rangle \mid \langle v \rangle \mid \langle V, V \rangle & \text{Module values} \\
 \mid \lambda^{\text{gn}} \alpha / m : \sigma. M \mid \lambda^{\text{ap}} \alpha / m : \sigma. M &
 \end{array}$$

The logical relation for the calculus is more complicated than the one in §2. Even so, the statement of the abstraction theorem for terms is similar to the one in §2.

Theorem 3 (Abstraction theorem [14]). *Suppose that $\vdash \Gamma \text{ ok}$. If $\Gamma \vdash e : \tau$, then $\Gamma \vdash e \sim e : \tau$.*

Modules and terms are interdependent, and Cray's theorem includes corresponding results for pure and for impure modules. We express security in terms of sealed modules, but our security proof only relies on the abstraction theorem for expressions.

Free theorem: TRNI for the module calculus. We present the idea of the encoding for the module calculus. (Formalization of the encoding can be found in §G.) To make the presentation easier to follow, in this section, we write examples in Standard ML (SML). These examples are checked with SML of New Jersey, version 110.96 [45].

For a policy \mathcal{P} , we construct the public view and the confidential view by using signatures containing type information of confidential inputs and their associated declassifiers. In particular, the signature for the confidential view is a transparent signature which exposes the concrete type of confidential input, while the signature for the public view is an opaque one which hides the type information of confidential inputs. For example, for the policy \mathcal{P}_{OE} (see Example 1), we have the following signatures, where `transOE` and `opaqueOE` are respectively the transparent signature for the confidential view and the opaque signature for the public view.

```

signature transOE = signature opaqueOE =
sig
  type t = int
  val x:t
  val f:t->int
end
sig
  type t
  val x:t
  val f:t->int
end

```

Different from §4, a program has only a module input which is of the transparent signature and contains all confidential inputs and their declassifiers. A program can use the input via the module variable \mathbf{m} . For example, for \mathcal{P}_{OE} , we have the program $\mathbf{m.f} \ \mathbf{m.x}$, which is corresponding to the program $x_f \ x$ in Example 5.

Using the result in §4, we define indistinguishability as an instantiation of the logical relation, and we say that a term e is $\text{TRNI}(\mathcal{P}, \tau)$ if on indistinguishable substitutions w.r.t. \mathcal{P} , it generates indistinguishable outputs at τ . By using the abstraction theorem 3 for terms, we obtain our main result.

Theorem 4. *If the type of e in the public view is τ , then e is $\text{TRNI}(\mathcal{P}, \tau)$.*

For the module calculus, when e is well-typed in the public view, e is also well-typed in the confidential view. Therefore, different from Theorem 2 which requires that e has no type variable, Theorem 4 simply requires that e is well-typed in the public view. Our example program $\mathbf{m.f} \ \mathbf{m.x}$ typechecks at \mathbf{int} , so by Theorem 4 it is $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$.

Usage of our approach. We can use our approach with ordinary ML implementations. In the case that the source programs are already parameterized by one module for their confidential inputs and their declassifiers, then there is no need to modify source programs at all.

For example, we consider **program** described below. Here \mathbf{M} is a module of the transparent signature **transOE**. By sealing this module with the opaque signature **opaqOE**, we get the module **opaqM**. Intuitively, **program** is $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$ since the declassifier \mathbf{f} is applied to the confidential input \mathbf{x} . We also come to the same conclusion from the fact that the type of this program is \mathbf{int} .

```
structure M = struct
  type t = int
  val x : t = 1
  val f : t -> int = fn x => x mod 2
end
structure opaqM :> opaqOE = M
val program : int = opaqM.f opaqM.x
```

So far our discussion is about open terms but the ML type checker only applies to closed terms. In the case that the client program is open (i.e. that it can receive any module of the transparent signature as an input, as in the program $\mathbf{m.f} \ \mathbf{m.x}$ presented above), in order to be able to type check it for a policy, we need to close it by putting in a closing context, which we call wrapper. For any program e and policy \mathcal{P} , the wrapper is written using a functor as shown below, where **opaqP** is the opaque signature for the public view of \mathcal{P} . Type τ is the type at which we want to check security of e . (The identifiers **program** and **wrapper** are arbitrary.)

```
functor wrapper (structure m: opaqP) =
  struct
    val program :  $\tau$  = e
  end
```


Note that e is unchanged.

We have proved that if the wrapper $wrap_{\mathcal{P}}(e)$ is of the signature from `opaque` to τ , then the type of e in the public view is τ . Therefore, from Theorem 4, e is TRNI at τ . For instance, for the policy \mathcal{P}_{OE} , we have that $wrap_{\mathcal{P}}(m.f \ m.x)$ is of the signature from `transOE` to `int` and hence, we infer that the type of `m.f m.x` in the public view is `int` and hence, `m.f m.x` is $TRNI(\mathcal{P}_{OE}, \text{int})$.

Extension. As in the case of the simple calculus, our encoding for ML can also be extended for policies where multiple inputs are declassified via a declassifier. Here, for illustration purpose, we present the encoding for a policy which is inspired by two-factor authentication.

Example 10. The policy \mathcal{P}_{Aut} involves two confidential passwords and two declassifiers `checking1` and `checking2` as below, where `input1` and `input2` are respectively the first input and the second input from a user. Notice that `checking2` takes a tuple of two passwords as its input.

```
fun checking1(password1:int) =
  if (password1 = input1) then 1 else 0
fun checking2(passwords:int*int) =
  if ((#1 passwords) = input1) then
    if ((#2 passwords) = input2) then 1 else 0
  else 2
```

We next construct the confidential view and the public view for the policy. To encode the requirement that two passwords can be declassified via `checking2`, we introduce a new variable `passwords` which is corresponding to the tuple of the two passwords, and only `passwords` can be declassified via `checking2`. The transparent signature for the confidential view of \mathcal{P}_{Aut} is below.

```
signature transAut = sig
  type t1 = int
  val password1:t1
  val checking1:t1->int
  type t2 = int
  val password2:t2
  type t3 = int * int
  val passwords:t3
  val checking2:t3 ->int
end
```

The signature `opaqueAut` for the public view is the same except the types `t1`, `t2`, and `t3` are opaque.

We have that the programs `m.checking2 m.passwords` and `m.checking1 m.password1`, where `m` is a module variable of the transparent signature `transAut`, have the type `int` in the public view. Hence both programs are $TRNI(\mathcal{P}_{Aut}, \text{int})$.

C Computation at multiple security levels: summary

The encodings in the preceding sections do not support computation and output at multiple levels. For example, consider a policy where x is a confidential input that can be declassified via f and we also want to do the computation $x + 1$ of which the result is at confidential level. Clearly, $x + 1$ is ill-typed in the public interface. To support computation at multiple levels we develop a monadic encoding inspired by DCC, and a public interface that represents policy for multiple levels.

To have an encoding that support multiple levels, we add universally quantified types $\forall\alpha.\tau$ to the language presented in §2 (already present in ML). In addition, to simplify the encoding, we add the unit type **unit**. W.r.t. these new types, we have new values: the unit value $\langle \rangle$ of **unit**, and values $\Lambda\alpha.e$ of $\forall\alpha.\tau$.

To facilitate the presentation of the idea of the encoding, we consider a lattice \mathcal{L} with three different levels L, M, H such that $L \sqsubset M \sqsubset H$. We also use a simple policy \mathcal{P} with three inputs hi, mi and li at resp. H, M and L , and hi can be declassified via $f : \mathbf{int} \rightarrow \mathbf{int}$ to M . (The encoding with an arbitrary finite lattice and policy is in §I.) For simplicity, we suppose that values on input and output channels are of **int** type.

To model multiple outputs we consider programs that return a tuple of values, one component for each output channel. To model channel access being associated with different security levels, the output values are wrapped, in the form $\lambda x : \mathbf{unit}.n$. To read such a value, an observer needs to provide an appropriate key. By giving x an abstract type corresponding to a security level, we can control access.

Similar to the previous sections, we assume that free variables in programs are their inputs, but now the values will be wrapped integers. Intuitively, a wrapped value v can be unwrapped by $unwrap\ k\ v$, where k is an appropriate key, and $unwrap\ k\ v$ can be implemented as the application of v on k (i.e. $v\ k$). Concretely, k will be the unit value $\langle \rangle$.

We further assume that programs are executed in a context where there are several output channels, each corresponding to a security level. A program will compute a tuple of wrapped values, where each element of the output tuple can be unwrapped by using an appropriate key and the unwrapped value is sent to the channel. In short, we assume the program of interest is executed in a context that wraps its inputs, and also unwraps each components of the output tuple and sends the value on the corresponding channel. This assumption is illustrated in the following pseudo program, where e is the program of interest, o is the computed tuple, Output.Channel_l is an output channel at l , k_l is a key to unwrap value at l , and π_l projects the output value for the output channel l .

```
let o = e in
  Output.ChannelL := unwrap kL (πL o)
  Output.ChannelM := unwrap kM (πM o)
  Output.ChannelH := unwrap kH (πH o)
```

The keys are not made directly available to e , which must manipulate its inputs via an interface described below.

Encoding. Different from the previous sections, we use type variables α_H, α_M and α_L as the types of keys for unwrapping wrapped values at H , M and L . This idea is similar to the idea in [27]. Different from [27], we do not translate DCC and we support declassification.

For an input at l that cannot be declassified (mi or li), its type in the public view is $\alpha_l \rightarrow \mathbf{int}$. For the input hi which can be declassified via f , we use another type variable (i.e. α_H^f) as the type of key to unwrapped values.⁹ Similar to the previous sections, we use α^f to encode number values at H . Therefore, the type of hi in the public view is $\alpha_H^f \rightarrow \alpha^f$. As we use **unit** as the type for key, in the confidential view the type of hi , mi , and li is **unit** $\rightarrow \mathbf{int}$.

As assumed above, a program computes an output which is a tuple of three wrapped values. Since we use type variables as keys to unwrap wrapped values, the type of outputs of programs we consider is $(\alpha_H \rightarrow \mathbf{int}) \times (\alpha_M \rightarrow \mathbf{int}) \times (\alpha_L \rightarrow \mathbf{int})$.

To support computing outputs at a level l , by using the idea of monad, we have interfaces \mathbf{cp}_l and \mathbf{wr}_l which are the bind and unit expressions for a monad. In addition, to support converting a wrapped value at l to l' (where $l \sqsubseteq l'$), we have interfaces $\mathbf{cvu}_l^{l'}$. To use hi in a computation at H , we have \mathbf{cv}_f . Similar to the previous sections, we have hi_f for the declassifier f . The types of there interfaces are described in Fig. 5.

$$\begin{aligned}
\Delta_{\mathcal{P}} &= \{\alpha_L, \alpha_M, \alpha_H, \alpha_H^f, \alpha^f\} \\
\Gamma_{\mathcal{P}} &= \{hi : \alpha_H^f \rightarrow \alpha^f, mi : \alpha_M \rightarrow \mathbf{int}, li : \alpha_L \rightarrow \mathbf{int}\} \cup \\
&\quad \{\mathbf{cp}_l : \forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \\
&\quad \quad \quad \rightarrow \alpha_l \rightarrow \beta_2 \mid l \in \mathcal{L}\} \cup \\
&\quad \{\mathbf{cvu}_l^{l'} : \forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta) \mid l, l' \in \mathcal{L} \wedge l \sqsubseteq l'\} \cup \\
&\quad \{\mathbf{wr}_l : \forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta \mid l \in \mathcal{L}\} \cup \\
&\quad \{hi_f : (\alpha_H^f \rightarrow \alpha^f) \rightarrow (\alpha_M \rightarrow \mathbf{int})\} \cup \\
&\quad \{\mathbf{cv}_f : (\alpha_H^f \rightarrow \alpha^f) \rightarrow (\alpha_H \rightarrow \mathbf{int})\}
\end{aligned}$$

Fig. 5. Contexts for \mathcal{P}

Example 11. We illustrate the idea of the encoding by writing a program that computes the triple $hi + li + 1$, $f \ hi$ and $li + 1$ at resp. H , M , and L .

⁹ If we use α_H instead, since this input can be declassified to M , the indistinguishability will be incorrect: all wrapped values at H are wrongly indistinguishable to observer M .

First, we will have e_1 that does the computation at L : $li + 1$. Let $plus_one = \lambda x : \mathbf{int}. x + 1$. In order to use \mathbf{cp}_L , we first wrap $plus_one$ by using \mathbf{wr}_L .

$$\mathbf{wrap_plus_one} = \lambda x : \mathbf{int}. \mathbf{wr}_L(\mathbf{plus_one} \ x)$$

Then e_1 is as below:

$$e_1 = \mathbf{cp}_L[\mathbf{int}][\mathbf{int}] \ li \ \mathbf{wrap_plus_one}$$

Next, we have e_2 that does the computation $hi + li$ at H . Let add be a function of the type $\mathbf{int} \rightarrow \mathbf{int} \rightarrow \mathbf{int}$. From add , we construct $\mathbf{wrap_addc}$ of the type $\mathbf{int} \rightarrow \alpha_H \rightarrow \mathbf{int}$, where c is a variable of the type \mathbf{int} .

$$\mathbf{wrap_addc} = \lambda y : \mathbf{int}. \mathbf{wr}_H(\mathbf{add} \ c \ y)$$

Then we have e_2 as below. Note that in order to use li in \mathbf{cp}_H , we need to convert li from level L to level H by using \mathbf{cvu}_L^H .

$$e_2 = \mathbf{cp}_H[\mathbf{int}][\mathbf{int}] \ hi \ (\lambda c : \mathbf{int}. (\mathbf{cp}_H[\mathbf{int}][\mathbf{int}] \ (\mathbf{cvu}_L^H \ li) \ \mathbf{wrap_addc}))$$

At this point, we can write the program e that computes $hi + li + 1$, $f \ hi$, and $li + 1$ at resp. H , M , and L .

$$e = (\lambda li : \alpha_L \rightarrow \mathbf{int}. \langle e_2, \langle \mathbf{hi}_f \ hi, li \rangle \rangle) \ e_1$$

The implementations of the defined interfaces are straightforward. For example, on a protected input of type $\mathbf{unit} \rightarrow \beta_1$ and a continuation of type $\beta_1 \rightarrow (\mathbf{unit} \rightarrow \beta_2)$, the implementation \mathbf{comp} of \mathbf{cp}_l first unfolds the protected input by applying it to the key $\langle \rangle$ and then applies the continuation on the result.

$$\mathbf{comp} = \lambda \beta_1, \beta_2. \lambda x : \mathbf{unit} \rightarrow \beta_1. \lambda f : \beta_1 \rightarrow (\mathbf{unit} \rightarrow \beta_2). f(x \ \langle \rangle)$$

The implementation of \mathbf{wr}_l is $\lambda \beta. \lambda x : \beta. \lambda _ : \mathbf{unit}. x$. The conversions $\mathbf{cvu}_l^{l'}$ are implemented by the identity function.

Indistinguishability. Different from §4, indistinguishability is defined for observer ζ ($\zeta \in \mathcal{L}$).¹⁰ The indistinguishability relations for ζ at type τ on values (denoted as $I_{\mathcal{P}}^{\zeta}[\tau]$) is defined as an instance of the logical relation with a careful choice of interpretations for α_l and α_H^f . The idea is that if the observer ζ cannot observe data at l (i.e. $l \not\sqsubseteq \zeta$), ζ does not have any key to unwrap these values and hence, all wrapped values at l are indistinguishable to ζ . Thus, α_l is interpreted as the empty relation for ζ . Otherwise, since ζ has key and can unwrap values, values wrapped at l are indistinguishable to ζ if they are equal and hence, α_l is interpreted as $\{\langle \rangle, \langle \rangle\}$ (note that the concrete type for key is \mathbf{unit}). Since wrapped numbers from hi are indistinguishable to observer ζ when they cannot

¹⁰ Following [12], we use ζ for observers.

be distinguish by the declassifier f , the interpretation of α_H^f for the observer M is $\{\langle\langle\rangle, \langle\rangle\rangle\}$.¹¹ By using the idea in the previous sections, based on $I_{\mathcal{P}}^{\zeta}[\tau]$, we define indistinguishability relations for ζ at type τ on terms (denoted as $I_{\mathcal{P}}^{\zeta}[\tau]^{ev}$).

Free theorem. We write ρ as an environment that maps type variables to its interpretations of form $\langle\tau_1, \tau_2, R\rangle$ and maps term variables to tuples of values. (This is similar to the formalization for §G.) We write ρ_L and ρ_R for the mappings that map every variable in the domain of ρ to respectively the first element and the second element of the tuple that ρ maps that variable to. The application of ρ_L (resp. ρ_R) to e is denoted by $\rho_L(e)$ (resp. $\rho_R(e)$) (this notations is similar to $\delta\gamma(e)$ in §2). We write $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$ to mean that ρ maps inputs to tuples of indistinguishable values.

By leveraging the abstraction theorem, we get the free theorem saying that a well-typed program e maps indistinguishable inputs to indistinguishable outputs.

Theorem 5. *If $\Delta_{\mathcal{P}}, \Gamma_{\mathcal{P}} \vdash e : \tau$, then for any $\zeta \in \mathcal{L}$ and $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$,*

$$\langle\rho_L(e), \rho_R(e)\rangle \in I_{\mathcal{P}}^{\zeta}[\tau]^{ev}.$$

We state it this way to avoid spelling out the definition of TRNI for this encoding. The encoding presented here can also be extended to support richer policies described in Remark 1.

Remark 2. Our encoding supports declassification while DCC does not. However, if we consider programs without declassification then DCC is more expressive since in our encoding, to use a wrapped value in a computation at l , this value must be wrapped at l' such that $l' \sqsubseteq l$. However, in DCC, this is not the case due to the definition of the “protected at” judgment in DCC: if type τ is already protected at l then so is $T_{l'}\tau$ for any l' . Therefore, data protected at l can be used in a computation protected at l' even when $l' \not\sqsubseteq l$. For example, we consider the encoding for a policy defined in a lattice with four levels \top, M_1, M_2, \perp where $\top \sqsupset M_i \sqsupset \perp$ but M_1 and M_2 are incomparable. We can have the following well-typed program in DCC (the program is written in the notations in [12]).

$$\text{bind } y = (\eta_{M_1} 1) \text{ in } \eta_{M_2}(\eta_{M_1}(y + 1))$$

In our encoding, this program can be rewritten as below, where $f : \mathbf{int} \rightarrow \alpha_{M_2} \rightarrow \alpha_{M_1} \rightarrow \mathbf{int}$ is from function $\lambda y : \mathbf{int}. y + 1$ (see a similar function in Example 11).

$$\text{cp}_{M_2}[\mathbf{int}][\mathbf{int}] (\text{wr}_{M_1} 1) f$$

This program is not well-typed in our encoding. This feature of DCC, allowing multiple layers of wrapping, is needed to encode state-passing programs (in particular, to encode the Volpano-Smith system for while programs) where low data is maintained unchanged through high computations. The feature seems unnecessary for functional programs.

¹¹ Therefore, if we used $\alpha_H \rightarrow \alpha_f$ for hi , all wrapped values from hi would be indistinguishable to the observer M since this observer do not have any key to open data at H that cannot be declassified (to observer M , the interpretation of α_H is empty).

D Proofs of Section 2

Lemma 1. Suppose that $\rho \in \text{Rel}(\delta_1, \delta_2)$ for some δ_1 and δ_2 . For $i \in \{1, 2\}$, it follows that:

- if $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_\rho$, then $\vdash v_i : \delta_i(\tau)$, and
- if $\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}$, then $\vdash e_i : \delta_i(\tau)$.

Proof. The second part of the lemma follows directly from rule FR-Term. We prove the first part of the lemma by induction on structure of τ .

Case 1: int. We consider $\langle v_1, v_2 \rangle \in \llbracket \text{int} \rrbracket_\rho$. From FR-Int, we have that $\vdash v_i : \text{int}$. Since $\delta_i(\text{int}) = \text{int}$, we have that $\vdash v_i : \delta_i(\text{int})$.

Case 2: α . We consider $\langle v_1, v_2 \rangle \in \llbracket \alpha \rrbracket_\rho$. From the FR-Var rule, $\langle v_1, v_2 \rangle \in \rho(\alpha) \in \text{Rel}(\delta_1(\alpha), \delta_2(\alpha))$. From the definition of $\text{Rel}(\delta_1(\alpha), \delta_2(\alpha))$, we have that $\vdash v_i : \delta_i(\alpha)$.

Case 3: $\tau_1 \times \tau_2$. We consider $\langle v_1, v_2 \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho$. We then have that $v_1 = \langle v_{11}, v_{12} \rangle$ for some v_{11} and v_{12} and $v_2 = \langle v_{21}, v_{22} \rangle$ for some v_{21} and v_{22} . From FR-Pair, it follows that $\langle v_{11}, v_{21} \rangle \in \llbracket \tau_1 \rrbracket_\rho$ and $\langle v_{12}, v_{22} \rangle \in \llbracket \tau_2 \rrbracket_\rho$. From IH (on τ_1 and τ_2), we have that $\vdash v_{11} : \delta_1(\tau_1)$, $\vdash v_{21} : \delta_2(\tau_1)$, $\vdash v_{12} : \delta_1(\tau_2)$, and $\vdash v_{22} : \delta_2(\tau_2)$. From FT-Pair, $\vdash \langle v_{11}, v_{12} \rangle : \delta_1(\tau_1) \times \delta_1(\tau_2)$ and $\vdash \langle v_{21}, v_{22} \rangle : \delta_2(\tau_1) \times \delta_2(\tau_2)$. Thus, $\vdash v_1 : \delta_1(\tau_1) \times \delta_1(\tau_2)$ and $\vdash v_2 : \delta_2(\tau_1) \times \delta_2(\tau_2)$. In other words, $\vdash v_1 : \delta_1(\tau_1 \times \tau_2)$ and $\vdash v_2 : \delta_2(\tau_1 \times \tau_2)$.

Case 4: $\tau_1 \rightarrow \tau_2$. We consider $\langle v_1, v_2 \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_\rho$. We now look at arbitrary $\langle v'_1, v'_2 \rangle \in \llbracket \tau_1 \rrbracket_\rho$. From FR-Fun, it follows that $\langle v_1 v'_1, v_2 v'_2 \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}$. From IH on τ_1 and the second part of the lemma on τ_2 , we have that $\vdash v'_i : \delta_i(\tau_1)$ and $\vdash v_i v'_i : \delta_i(\tau_2)$ (for $i \in \{1, 2\}$). From FT-App, we have that $\vdash v_i : \delta_i(\tau_1 \rightarrow \tau_2)$.

Theorem 1. If $\Delta, \Gamma \vdash e : \tau$, then $\Delta, \Gamma \vdash e \sim e : \tau$.

Proof. We prove the theorem by induction on typing derivation.

Case 1: Rule FT-Int.

$$\overline{\vdash n : \text{int}}$$

From FR-Int, we have that $\langle n, n \rangle \in \llbracket \text{int} \rrbracket_\rho$. Therefore, from FR-Term, it follows that $\langle n, n \rangle \in \llbracket \text{int} \rrbracket_\rho^{\text{ev}}$. Hence, $\langle \delta_1 \gamma_1(n), \delta_2 \gamma_2(n) \rangle \in \llbracket \text{int} \rrbracket_\rho^{\text{ev}}$. The proof is closed for this case.

Case 2: Rule FT-Var.

$$\frac{x : \tau \in \Gamma}{\Delta, \Gamma \vdash x : \tau}$$

Since $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma \rrbracket_\rho$ and $x \in \text{dom}(\gamma_1) = \text{dom}(\gamma_2)$, we have that $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \Gamma(x) \rrbracket_\rho$, and hence $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \Gamma(x) \rrbracket_\rho^{\text{ev}}$.

Since there is no type variable in x , we have that $\delta_1 \gamma_1(x) = \gamma_1(x)$ and $\delta_2 \gamma_2(x) = \gamma_2(x)$. As proven above, $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \Gamma(x) \rrbracket_\rho^{\text{ev}}$. Thus, we have that $\langle \delta_1 \gamma_1(x), \delta_2 \gamma_2(x) \rangle \in \llbracket \Gamma(x) \rrbracket_\rho^{\text{ev}}$. The proof is closed for this case.

Case 3: Rule FT-Pair.

$$\frac{\Delta, \Gamma \vdash e_1 : \tau_1 \quad \Delta, \Gamma \vdash e_2 : \tau_2}{\Delta, \Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2}$$

From IH, it follows that

- $\langle \delta_1 \gamma_1(e_1), \delta_2 \gamma_2(e_1) \rangle \in \llbracket \tau_1 \rrbracket_\rho^{\text{ev}}$, and
- $\langle \delta_1 \gamma_1(e_2), \delta_2 \gamma_2(e_2) \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}$.

From the FR-Term, we have that

- $\langle v_{11}, v_{12} \rangle \in \llbracket \tau_1 \rrbracket_\rho$, where $\langle \delta_1 \gamma_1(e_1), \delta_2 \gamma_2(e_1) \rangle \rightarrow^* \langle v_{11}, v_{12} \rangle$, and
- $\langle v_{21}, v_{22} \rangle \in \llbracket \tau_2 \rrbracket_\rho$, where $\langle \delta_1 \gamma_1(e_2), \delta_2 \gamma_2(e_2) \rangle \rightarrow^* \langle v_{21}, v_{22} \rangle$.

From FR-Pair, it follows that $\langle \langle v_{11}, v_{21} \rangle, \langle v_{12}, v_{22} \rangle \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho$. From FR-Term, we have that

$$\langle \langle \delta_1 \gamma_1(e_1), \delta_1 \gamma_1(e_2) \rangle, \langle \delta_2 \gamma_2(e_1), \delta_2 \gamma_2(e_2) \rangle \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho^{\text{ev}}.$$

Thus, $\langle \delta_1 \gamma_1 \langle e_1, e_2 \rangle, \delta_2 \gamma_2 \langle e_1, e_2 \rangle \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho^{\text{ev}}$. This case is closed.

Case 4: rule FT-Prj.

$$\frac{\Delta, \Gamma \vdash e : \tau_1 \times \tau_2}{\Delta, \Gamma \vdash \pi_i e : \tau_i}$$

From IH, we have that $\langle \delta_1 \gamma_1(e), \delta_2 \gamma_2(e) \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho^{\text{ev}}$. Thus, $\langle \langle v_{11}, v_{21} \rangle, \langle v_{12}, v_{22} \rangle \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho$, where $\delta_1 \gamma_1(e) \rightarrow^* \langle v_{11}, v_{21} \rangle$ and $\delta_2 \gamma_2(e) \rightarrow^* \langle v_{12}, v_{22} \rangle$. From FR-Pair, we have that $\langle v_{11}, v_{12} \rangle \in \llbracket \tau_1 \rrbracket_\rho$ and $\langle v_{21}, v_{22} \rangle \in \llbracket \tau_2 \rrbracket_\rho$.

Since $\delta_1 \gamma_1(e) \rightarrow^* \langle v_{11}, v_{21} \rangle$ and $\delta_2 \gamma_2(e) \rightarrow^* \langle v_{12}, v_{22} \rangle$

- $\pi_1 \delta_1 \gamma_1(e) \rightarrow^* v_{11}$,
- $\pi_2 \delta_1 \gamma_1(e) \rightarrow^* v_{12}$,
- $\pi_1 \delta_2 \gamma_2(e) \rightarrow^* v_{12}$,
- $\pi_2 \delta_2 \gamma_2(e) \rightarrow^* v_{22}$.

Thus,

- $\delta_1 \gamma_1(\pi_1 e) \rightarrow^* v_{11}$,
- $\delta_1 \gamma_1(\pi_2 e) \rightarrow^* v_{12}$,
- $\delta_2 \gamma_2(\pi_1 e) \rightarrow^* v_{12}$,
- $\delta_2 \gamma_2(\pi_2 e) \rightarrow^* v_{22}$.

Thus,

- $\langle \delta_1 \gamma_1(\pi_1 e), \delta_2 \gamma_2(\pi_1 e) \rangle \rightarrow^* \langle v_{11}, v_{12} \rangle$ and
- $\langle \delta_1 \gamma_1(\pi_2 e), \delta_2 \gamma_2(\pi_2 e) \rangle \rightarrow^* \langle v_{12}, v_{22} \rangle$.

As proven above $\langle v_{11}, v_{12} \rangle \in \llbracket \tau_1 \rrbracket_\rho$ and $\langle v_{21}, v_{22} \rangle \in \llbracket \tau_2 \rrbracket_\rho$. From FR-Term, we have that $\langle \delta_1 \gamma_1(\pi_1 e), \delta_2 \gamma_2(\pi_1 e) \rangle \in \llbracket \tau_1 \rrbracket_\rho^{\text{ev}}$ and $\langle \delta_1 \gamma_1(\pi_2 e), \delta_2 \gamma_2(\pi_2 e) \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}$. This case is closed.

Case 5: rule FT-Fun.

$$\frac{\Delta, \Gamma, x : \tau_1 \vdash e : \tau_2}{\Delta, \Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2}$$

From IH, we have that $\langle \delta_1 \gamma_1[x \mapsto v_1](e), \delta_2 \gamma_2[x \mapsto v_2](e) \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}$, where $\langle v_1, v_2 \rangle \in \llbracket \tau_1 \rrbracket_\rho^{\text{ev}}$. Hence, we have that $\langle \delta_1 \gamma_1(e[x \mapsto v_1]), \delta_2 \gamma_2(e[x \mapsto v_2]) \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}$. From the semantics of the language and the FR-Term rule, it follows that

$$\langle \delta_1 \gamma_1((\lambda x : \tau_1. e) v_1), \delta_2 \gamma_2((\lambda x : \tau_1. e) v_2) \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}.$$

Since $\langle v_1, v_2 \rangle$ are arbitrary, from FR-Fun, we have that $\langle \delta_1 \gamma_1(\lambda x : \tau_1. e), \delta_2 \gamma_2(\lambda x : \tau_1. e) \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_\rho$, and hence $\langle \delta_1 \gamma_1(\lambda x : \tau_1. e), \delta_2 \gamma_2(\lambda x : \tau_1. e) \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_\rho^{\text{ev}}$. The proof is closed for this case.

Case 6: rule FT-App.

$$\frac{\Delta, \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Delta, \Gamma \vdash e_2 : \tau_1}{\Delta, \Gamma \vdash e_1 e_2 : \tau_2}$$

The proof follows from IH and rule FR-Fun.

E Proofs of Section 4

Lemma 2. $\rho_{\mathcal{P}} \in \text{Rel}(\delta_{\mathcal{P}}, \delta_{\mathcal{P}})$.

Proof. We need to prove that for any type variable α , if $\langle v_1, v_2 \rangle \in \rho_{\mathcal{P}}(\alpha)$, then $\vdash v_i : \delta_{\mathcal{P}}(\alpha)$ and hence $\vdash v_i : \mathbf{int}$ according to the definition (1). This follows directly from the definition of $\rho_{\mathcal{P}}$ and rules Eq-Var1, Eq-Var2, and Eq-Var3.

Lemma 5. Suppose that $\vdash v : \tau$. It follows that:

- if τ is \mathbf{int} , v is n for some n ,
- if τ is $\tau_1 \rightarrow \tau_2$, v is $\lambda x : \tau_1. e$ for some e ,
- if τ is $\tau_1 \times \tau_2$, it is $\langle v_1, v_2 \rangle$ for some v_1 and v_2 .

Proof. We prove the case of \mathbf{int} first by case analysis on typing rules. The FT-Int rule gives us the desired result. The other rules cannot be instantiated with an expression that is a value and of the \mathbf{int} type.

We prove the case of $\tau_1 \rightarrow \tau_2$ by case analysis on typing rules. The FT-Fun rule gives us the desired result. The other rules cannot be instantiated with an expression that is a value and of the $\tau_1 \rightarrow \tau_2$ type.

The proof for the case $\tau_1 \times \tau_2$ is similar.

Lemma 3. It follows that

- $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_{\rho_{\mathcal{P}}}$ iff $\langle v_1, v_2 \rangle \in \mathcal{I}_V \llbracket \tau \rrbracket$, and
- $\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_{\rho_{\mathcal{P}}}^{\text{ev}}$ iff $\langle e_1, e_2 \rangle \in \mathcal{I}_E \llbracket \tau \rrbracket$.

Proof. We prove the lemma by induction on the structure of τ .

Case 1: int. We consider $\mathcal{I}_V \llbracket \mathbf{int} \rrbracket$ and $\llbracket \mathbf{int} \rrbracket_{\rho_{\mathcal{P}}}$. We have that $\langle n, n \rangle \in \llbracket \mathbf{int} \rrbracket_{\rho_{\mathcal{P}}}$ iff $\langle n, n \rangle \in \mathcal{I}_V \llbracket \mathbf{int} \rrbracket$.

We consider $\mathcal{I}_E \llbracket \mathbf{int} \rrbracket$ and $\llbracket \mathbf{int} \rrbracket_{\rho_{\mathcal{P}}}^{\text{ev}}$. We consider $\langle e_1, e_2 \rangle$.

- If $\langle e_1, e_2 \rangle \in \mathcal{I}_V[\mathbf{int}]$, from Eq-Term and Eq-Int, there exists n s.t. $e_i \rightarrow^* n$. From FR-Int and FR-Term, $\langle e_1, e_2 \rangle \in [\mathbf{int}]_{\rho_P}^{\text{ev}}$.
- If $\langle e_1, e_2 \rangle \in [\mathbf{int}]_{\rho_P}^{\text{ev}}$, from FR-Term and FR-Int, there exists n s.t. $e_i \rightarrow^* n$. From Eq-Int and Eq-Term, $\langle e_1, e_2 \rangle \in \mathcal{I}_E[\mathbf{int}]$.

Case 2: α_x . First we consider $\mathcal{I}_V[\alpha_x]$ and $[\alpha_x]_{\rho_P}$. From the definition of ρ_P and the FR-Var rule, $\langle v_1, v_2 \rangle \in [\alpha_x]_{\rho_P}$ iff $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\alpha_x]$.

Case 3: α_f . The proof is similar to the one of Case 2.

Case 4: $\tau_1 \times \tau_2$. We first consider $\mathcal{I}_V[\tau_1 \times \tau_2]$ and $[\tau_1 \times \tau_2]_{\rho_P}$.

- Suppose that $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \times \tau_2]$. From the definition of indistinguishability, we have that $\vdash v_i : \delta_P(\tau_1 \times \tau_2)$. From Lemma 5, $v_1 = \langle v_{11}, v_{12} \rangle$ and $v_2 = \langle v_{21}, v_{22} \rangle$. Thus, we have that $\langle \langle v_{11}, v_{12} \rangle, \langle v_{21}, v_{22} \rangle \rangle \in \mathcal{I}_V[\tau_1 \times \tau_2]$. From the Eq-Pair rule, we have that $\langle v_{11}, v_{21} \rangle \in \mathcal{I}_V[\tau_1]$ and $\langle v_{12}, v_{22} \rangle \in \mathcal{I}_V[\tau_2]$. From IH on τ_1 and τ_2 , we have that $\langle v_{11}, v_{21} \rangle \in [\tau_1]_{\rho_P}$ and $\langle v_{12}, v_{22} \rangle \in [\tau_2]_{\rho_P}$. From the FR-Pair, it follows that $\langle \langle v_{11}, v_{12} \rangle, \langle v_{21}, v_{22} \rangle \rangle \in [\tau_1 \times \tau_2]_{\rho_P}$.
- Suppose that $\langle v_1, v_2 \rangle \in [\tau_1 \times \tau_2]_{\rho_P}$. From Lemma 1, $\vdash v_i : \delta_P(\tau_1 \times \tau_2)$. Thus, we have that $\langle \langle v_{11}, v_{12} \rangle, \langle v_{21}, v_{22} \rangle \rangle \in [\tau_1 \times \tau_2]_{\rho_P}$. From the FR-Pair rule, we have that $\langle v_{11}, v_{21} \rangle \in [\tau_1]_{\rho_P}$ and $\langle v_{12}, v_{22} \rangle \in [\tau_2]_{\rho_P}$. From IH on τ_1 and τ_2 , we have that $\langle v_{11}, v_{21} \rangle \in \mathcal{I}_V[\tau_1]$ and $\langle v_{12}, v_{22} \rangle \in \mathcal{I}_V[\tau_2]$. From the Eq-Pair, it follows that $\langle \langle v_{11}, v_{12} \rangle, \langle v_{21}, v_{22} \rangle \rangle \in \mathcal{I}_V[\tau_1 \times \tau_2]$.

We now consider $\mathcal{I}_E[\tau_1 \times \tau_2]$ and $[\tau_1 \times \tau_2]_{\rho_P}^{\text{ev}}$.

- Suppose that $\langle e_1, e_2 \rangle \in \mathcal{I}_E[\tau_1 \times \tau_2]$. From Eq-Term, we have that $e_i \rightarrow^* v_i$ for some v_i and $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \times \tau_2]$. As proven above, we have that $\langle v_1, v_2 \rangle \in [\tau_1 \times \tau_2]_{\rho_P}$. From FR-Term, $\langle e_1, e_2 \rangle \in [\tau_1 \times \tau_2]_{\rho_P}^{\text{ev}}$.
- Suppose that $\langle e_1, e_2 \rangle \in [\tau_1 \times \tau_2]_{\rho_P}^{\text{ev}}$. From FR-Term, we have that $e_i \rightarrow^* v_i$ for some v_i and $\langle v_1, v_2 \rangle \in [\tau_1 \times \tau_2]_{\rho_P}$. As proven above, we have that $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \times \tau_2]$. From Eq-Term, $\langle e_1, e_2 \rangle \in \mathcal{I}_E[\tau_1 \times \tau_2]$.

Case 5: $\tau_1 \rightarrow \tau_2$. We first consider $\mathcal{I}_V[\tau_1 \rightarrow \tau_2]$ and $[\tau_1 \rightarrow \tau_2]_{\rho_P}$.

- Suppose $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \rightarrow \tau_2]$. We need to prove that for any $\langle v'_1, v'_2 \rangle \in [\tau_1]_{\rho_P}$, $\langle v_1 v'_1, v_2 v'_2 \rangle \in [\tau_2]_{\rho_P}^{\text{ev}}$. Since $\langle v'_1, v'_2 \rangle \in [\tau_1]_{\rho_P}$, from IH on τ_1 , we have that $\langle v'_1, v'_2 \rangle \in \mathcal{I}_V[\tau_1]$. Since $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \rightarrow \tau_2]$, from Eq-Fun, we have that $\langle v_1 v'_1, v_2 v'_2 \rangle \in \mathcal{I}_E[\tau_2]$. From IH on τ_2 , $\langle v_1 v'_1, v_2 v'_2 \rangle \in [\tau_2]_{\rho_P}^{\text{ev}}$.
- Suppose that $\langle v_1, v_2 \rangle \in [\tau_1 \rightarrow \tau_2]_{\rho_P}$. We need to prove that for any $\langle v'_1, v'_2 \rangle \in \mathcal{I}_V[\tau_1]$, $\langle v_1 v'_1, v_2 v'_2 \rangle \in \mathcal{I}_E[\tau_2]$. Since $\langle v'_1, v'_2 \rangle \in \mathcal{I}_V[\tau_1]$, from IH on τ_1 , we have that $\langle v'_1, v'_2 \rangle \in [\tau_1]_{\rho_P}$. Since $\langle v_1, v_2 \rangle \in [\tau_1 \rightarrow \tau_2]_{\rho_P}$, from Eq-Fun, we have that $\langle v_1 v'_1, v_2 v'_2 \rangle \in [\tau_2]_{\rho_P}^{\text{ev}}$. From IH on τ_2 , $\langle v_1 v'_1, v_2 v'_2 \rangle \in \mathcal{I}_E[\tau_2]$.

We now consider $\mathcal{I}_E[\tau_1 \rightarrow \tau_2]$ and $[\tau_1 \rightarrow \tau_2]_{\rho_P}^{\text{ev}}$.

- Suppose that $\langle e_1, e_2 \rangle \in \mathcal{I}_E[\tau_1 \rightarrow \tau_2]$. From Eq-Term, $e_i \rightarrow^* v_i$ for some v_i and $\langle v_1, v_2 \rangle \in \mathcal{I}_V[\tau_1 \rightarrow \tau_2]$. As proven above, we have that $\langle v_1, v_2 \rangle \in [\tau_1 \rightarrow \tau_2]_{\rho_P}$. Thus, $\langle e_1, e_2 \rangle \in [\tau_1 \rightarrow \tau_2]_{\rho_P}^{\text{ev}}$.

- Suppose that $\langle e_1, e_2 \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_{\rho_P}^{\text{ev}}$. From FR-Term, $e_i \rightarrow^* v_i$ for some v_i and $\langle v_1, v_2 \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_{\rho_P}$. As proven above, we have that $\langle v_1, v_2 \rangle \in \mathcal{I}_V \llbracket \tau_1 \rightarrow \tau_2 \rrbracket$. Thus, $\langle e_1, e_2 \rangle \in \mathcal{I}_E \llbracket \tau_1 \rightarrow \tau_2 \rrbracket$.

Lemma 4. If $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V \llbracket \mathcal{P} \rrbracket$, then $\langle \gamma_1, \gamma_2 \rangle \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket_{\rho_P}$.

Proof. We first prove that $\text{dom}(\gamma_1) = \text{dom}(\gamma_2) = \text{dom}(\Gamma_P^{\mathcal{P}})$. This is directly from the definition of $\langle \gamma_1, \gamma_2 \rangle \in \mathcal{I}_V \llbracket \mathcal{P} \rrbracket$.

We now need to prove that $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket_{\rho_P}$ for all x . From the construction of $\Gamma_P^{\mathcal{P}}$, we have the following cases.

Case 1: $\Gamma_P^{\mathcal{P}}(x) = \alpha_x$. From the assumption, we have that

$$\langle \gamma_1(x), \gamma_2(x) \rangle \in \mathcal{I}_V \llbracket \alpha_x \rrbracket.$$

From Lemma 3, it follows that $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \alpha_x \rrbracket_{\rho_P}$.

Case 2: $\Gamma_P^{\mathcal{P}}(x) = \alpha_f$. From the assumption, we have that

$$\langle \gamma_1(x), \gamma_2(x) \rangle \in \mathcal{I}_V \llbracket \alpha_f \rrbracket.$$

From Lemma 3, it follows that $\langle \gamma_1(x), \gamma_2(x) \rangle \in \llbracket \alpha_f \rrbracket_{\rho_P}$.

Case 3: $\Gamma_P^{\mathcal{P}}(x_f) = \alpha_f \rightarrow \tau$. We need to prove that $\langle f, f \rangle \in \llbracket \alpha_f \rightarrow \tau \rrbracket_{\rho_P}$, where $\vdash f : \mathbf{int} \rightarrow \tau$ for some τ s.t. $\vdash \tau$. From FR-Fun, we need to prove that for any $\langle v_1, v_2 \rangle \in \llbracket \alpha_f \rrbracket_{\rho_P}$, $\langle f v_1, f v_2 \rangle \in \llbracket \tau \rrbracket_{\rho_P}^{\text{ev}}$.

We consider an arbitrary $\langle v_1, v_2 \rangle \in \llbracket \alpha_f \rrbracket_{\rho_P}$. From the Eq-Var2 rule, we have that $\langle f v_1, f v_2 \rangle \in \mathcal{I}_E \llbracket \tau \rrbracket$. From Lemma 3, $\langle f v_1, f v_2 \rangle \in \llbracket \tau \rrbracket_{\rho_P}^{\text{ev}}$.

Case 4: $\Gamma_P^{\mathcal{P}}(x_a) = \alpha_{f \circ a} \rightarrow \alpha_f$. We need to prove that $\langle a, a \rangle \in \llbracket \alpha_{f \circ a} \rightarrow \alpha_f \rrbracket_{\rho_P}$, where $\vdash a : \mathbf{int} \rightarrow \mathbf{int}$. From FR-Fun, we need to prove that for any $\langle v_1, v_2 \rangle \in \llbracket \alpha_{f \circ a} \rrbracket_{\rho_P}$, $\langle a v_1, a v_2 \rangle \in \llbracket \alpha_f \rrbracket_{\rho_P}^{\text{ev}}$.

We consider an arbitrary $\langle v_1, v_2 \rangle \in \llbracket \alpha_{f \circ a} \rrbracket_{\rho_P}$. From the Eq-Var3 rule, we have that $\langle a v_1, a v_2 \rangle \in \mathcal{I}_E \llbracket \alpha_f \rrbracket$. From Lemma 3, $\langle a v_1, a v_2 \rangle \in \llbracket \alpha_f \rrbracket_{\rho_P}^{\text{ev}}$.

F Module Calculus

F.1 Syntax and semantics

This section presents a module calculus, essentially the same as that of Crary [14] except that we add **int** for integers. Crary's calculus is adapted from Dreyer's thesis [19], and the reader should consult these references for explanations and motivation. As described in §B, the calculus has static expressions: kinds (k), constructors (c) and signatures (σ), and dynamic expressions: terms (e) and modules (M). The syntax is in Fig. 4.

Kinds and constructors. The unit kind 1 has only the unit constructor \star . The kind \mathbf{T} have base types that can be used to classify terms. The singleton kind $\mathbf{S}(c)$ (where c is of the base kind) has constructors that are definitionally equivalent to c . In addition, we have higher kinds: dependent functions $\Pi \alpha : k_1.k_2$ and dependent pairs $\Sigma \alpha : k_1.k_2$. A constructor c of the kind $\Sigma \alpha : k_1.k_2$ has pairs

of constructors where the first component $\pi_1 c$ is of the kind k_1 and the second component $\pi_2 c$ is of the kind $k_2[\alpha \mapsto \pi_1 c]$. A constructor c of the kind $\Pi \alpha : k_1.k_2$ takes a constructor c' of kind k_1 as a parameter and returns a constructor of the kind $k_2[\alpha \mapsto c']$. When α does not appear free in k_2 , we write $k_1 \rightarrow k_2$ instead of $\Pi \alpha : k_1.k_2$ and $k_1 \times k_2$ instead of $\Sigma \alpha : k_1.k_2$. We use the metavariable τ for constructors that are types (i.e. of the kind \mathbf{T}).

Terms and modules. The syntax for terms is standard. Modules can be unit module, static atomic module, dynamic atomic module, generative functor, applicative functor, application, pair, projection, unpack, term binding, module binding, and sealing. Applications of generative functors and applicative functors are syntactically distinguished.

Abstraction is introduced by *sealing*: in the module $M :> \sigma$, access to the component M is limited to the interface σ . A term is extracted from a module $\langle e \rangle$ by $\text{Ext } \langle e \rangle$. To extract the static part of a module, we have the operation $\Gamma \vdash \text{Fst}(M) \gg c$ meaning that the static part of M is c in Γ . When the context is empty, we write $\text{Fst}(M)$ for c where $\vdash \text{Fst}(M) \gg c$.

Notice that every module variable is associated with a constructor variable that represents its static part [14]. The relation is maintained by twinned variables: $\alpha/m : \sigma$ meaning that m has signature σ and its static part is α which is of the kind $\text{Fst}(\sigma)$, where for any signature σ , $\text{Fst}(\sigma)$ extracts the information about kind from σ . Whenever $m : \sigma$ and $\text{Fst}(m) \gg c$, it follows that c is of the kind $\text{Fst}(\sigma)$.

$$\begin{array}{c}
\frac{\alpha/m \in \text{dom}(\Gamma)}{\Gamma \vdash \text{Fst}(m) \gg \alpha} \quad \frac{}{\Gamma \vdash \text{Fst}(\star) \gg \star} \quad \frac{}{\Gamma \vdash \text{Fst}(\langle c \rangle) \gg c} \quad \frac{}{\Gamma \vdash \text{Fst}(\langle e \rangle) \gg \star} \\
\\
\frac{}{\Gamma \vdash \text{Fst}(\lambda^{\text{gn}} \alpha/m : \sigma.M) \gg \star} \quad \frac{\Gamma, \alpha/m : \sigma \vdash \text{Fst}(M) \gg c}{\Gamma \vdash \text{Fst}(\lambda^{\text{ap}} \alpha/m : \sigma.M) \gg \lambda \alpha : \text{Fst}(\sigma).c} \\
\\
\frac{\Gamma \vdash \text{Fst}(M_1) \gg c_1 \quad \Gamma \vdash \text{Fst}(M_2) \gg c_2}{\Gamma \vdash \text{Fst}(M_1 \cdot M_2) \gg c_1 c_2} \\
\\
\frac{\Gamma \vdash \text{Fst}(M_1) \gg c_1 \quad \Gamma \vdash \text{Fst}(M_2) \gg c_2}{\Gamma \vdash \text{Fst}(\langle M_1, M_2 \rangle) \gg \langle c_1, c_2 \rangle} \quad \frac{\Gamma \vdash \text{Fst}(M) \gg c}{\Gamma \vdash \text{Fst}(\pi_i M) \gg \pi_i c} \\
\\
\frac{\Gamma \vdash \text{Fst}(M) \gg c}{\Gamma \vdash \text{Fst}(\text{let } x = e \text{ in } M) \gg c}
\end{array}$$

Fig. 6. Extracting constructor information from modules

Signature. Signatures include unit signature, atomic kind signature, atomic type signature, signatures for generative functors, applicative functors and pairs. Since

$$\begin{array}{ll}
\text{Fst}(1) \triangleq 1 & \text{Fst}(\Pi^{\text{gn}} \alpha : \sigma_1.\sigma_2) \triangleq 1 \\
\text{Fst}(\langle k \rangle) \triangleq k & \text{Fst}(\Pi^{\text{ap}} \alpha : \sigma_1.\sigma_2) \triangleq \Pi \alpha : \text{Fst}(\sigma_1).\text{Fst}(\sigma_2) \\
\text{Fst}(\langle \tau \rangle) \triangleq 1 & \text{Fst}(\Sigma \alpha : \sigma_1.\sigma_2) \triangleq \Sigma \alpha : \text{Fst}(\sigma_1).\text{Fst}(\sigma_2)
\end{array}$$

Fig. 7. Extracting kind information from signatures

a module does not appear in static part of a signature, we have only α in dependent signatures (instead of twinned variables, e.g. α/m , as in the case of modules). In the binding $\alpha : \sigma$ within a dependent signature, α corresponds to the static part of some module of the signature σ . Thus, α has the kind $\text{Fst}(\sigma)$.

As described in [19], a signature σ is *transparent* when it exposes the implementation of the static part of modules of σ . A signature σ is *opaque* when it hides some information about the static part of modules of σ .

Example 12. We consider the following module and signatures. Suppose that $f = \lambda x : \mathbf{int}.e$ for some e which is a closed function of the type $\mathbf{int} \rightarrow \tau_f$ for some closed τ_f .

| | | |
|--|---|---|
| <pre> structure M = struct type t = int val x:t = 0 val f:t->int = ... end </pre> | <pre> signature σ_T = sig type t = int val x:int val f:int -> τ_f end </pre> | <pre> signature σ_O = sig type t val x:t val f:t -> τ_f end </pre> |
|--|---|---|

In the module calculus, M is $\langle \langle \mathbf{int} \rangle, \langle \langle 0 \rangle, \langle \lambda x : \mathbf{int}.e \rangle \rangle \rangle$. Using abbreviations, σ_T is $\langle \langle S(\mathbf{int}) \rangle, \langle \langle \mathbf{int} \rangle, \langle \mathbf{int} \rightarrow \tau_f \rangle \rangle \rangle$ and σ_O is $\Sigma \alpha : \langle \mathbf{T} \rangle. \langle \langle \alpha \rangle, \langle \alpha \rightarrow \tau_f \rangle \rangle$.¹² The signature σ_T is a transparent signature of M since σ_T exposes the information of the static part of M , as $\langle S(\mathbf{int}) \rangle$. The signature σ_O is an opaque signature of M since σ_O hides the information of the static part of M , as $\langle \mathbf{T} \rangle$.

Static semantics. The judgment forms in the static semantics are described in Figure 8. W.r.t. the static semantics, for the signatures described in Example 12, it follows that the transparent signature σ_T is a sub-signature of the opaque signature σ_O .

¹² Expanding abbreviations, σ_T is $\Sigma \alpha : \langle S(\mathbf{int}) \rangle. \Sigma \beta : \langle \mathbf{int} \rangle. \langle \mathbf{int} \rightarrow \tau_f \rangle$ and σ_O is $\Sigma \alpha : \langle \mathbf{T} \rangle. \Sigma \beta : \langle \alpha \rangle. \langle \alpha \rightarrow \tau_f \rangle$.

| | | |
|---|---------------------------|----------|
| $\vdash \Gamma \text{ ok}$ | Well-formed context | (Fig 9) |
| $\Gamma \vdash k : \text{kind}$ | Well-formed kind | (Fig 10) |
| $\Gamma \vdash k_1 \equiv k_2 : \text{kind}$ | Kind equivalence | (Fig 11) |
| $\Gamma \vdash k_1 \leq k_2 : \text{kind}$ | Subkinding | (Fig 12) |
| $\Gamma \vdash c : k$ | Well-formed constructor | (Fig 13) |
| $\Gamma \vdash c_1 \equiv c_2 : \text{kind}$ | Constructor equivalence | (Fig 14) |
| $\Gamma \vdash e : \tau$ | Well-typed term | (Fig 15) |
| $\Gamma \vdash \sigma : \text{sig}$ | Well-formed signature | (Fig 16) |
| $\Gamma \vdash \sigma_1 \equiv \sigma_2 : \text{sig}$ | Equivalence signature | (Fig 17) |
| $\Gamma \vdash \sigma_1 \leq \sigma_2 : \text{sig}$ | Subsignature | (Fig 18) |
| $\Gamma \vdash_{\text{P}} M : \sigma$ | Pure well-formed module | (Fig 19) |
| $\Gamma \vdash_{\text{I}} M : \sigma$ | Impure well-formed module | (Fig 19) |

Fig. 8. Judgment forms in the static semantics

$$\begin{array}{c}
 \text{WF_NIL} \frac{}{\vdash . \text{ ok}} \quad \text{WF_CN} \frac{\vdash \Gamma \text{ ok} \quad \Gamma \vdash k : \text{kind}}{\vdash \Gamma, \alpha : k \text{ ok}} \quad \text{WF_TM} \frac{\vdash \Gamma \text{ ok} \quad \Gamma \vdash \tau : \mathbb{T}}{\vdash \Gamma, x : \tau \text{ ok}} \\
 \\
 \text{WF_MD} \frac{\vdash \Gamma \text{ ok} \quad \Gamma \vdash \sigma : \text{sig}}{\vdash \Gamma, \alpha/m : \sigma \text{ ok}}
 \end{array}$$

Fig. 9. Well-formed context $\vdash \Gamma \text{ ok}$

$$\begin{array}{c}
 \text{OFK_TYPE} \frac{}{\Gamma \vdash \mathbb{T} : \text{kind}} \quad \text{OFK_SING} \frac{\Gamma \vdash c : \mathbb{T}}{\Gamma \vdash S(c) : \text{kind}} \quad \text{OFK_ONE} \frac{}{\Gamma \vdash 1 : \text{kind}} \\
 \\
 \text{OFK_PI} \frac{\Gamma \vdash k_1 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash \Pi \alpha : k_1.k_2 : \text{kind}} \\
 \\
 \text{OFK_SIGMA} \frac{\Gamma \vdash k_1 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash \Sigma \alpha : k_1.k_2 : \text{kind}}
 \end{array}$$

Fig. 10. Well-formed kind $\Gamma \vdash k : \text{kind}$

$$\begin{array}{c}
 \text{EQK_REFL} \frac{\Gamma \vdash k : \text{kind}}{\Gamma \vdash k \equiv k : \text{kind}} \qquad \text{EQK_SYMM} \frac{\Gamma \vdash k_1 \equiv k_2 : \text{kind}}{\Gamma \vdash k_2 \equiv k_1 : \text{kind}} \\
 \\
 \text{EQK_TRANS} \frac{\Gamma \vdash k_1 \equiv k_2 : \text{kind} \quad \Gamma \vdash k_2 \equiv k_3 : \text{kind}}{\Gamma \vdash k_1 \equiv k_3 : \text{kind}} \\
 \\
 \text{EQK_SING} \frac{\Gamma \vdash c_1 \equiv c_2 : \mathbb{T}}{\Gamma \vdash S(c_1) \equiv S(c_2) : \text{kind}} \\
 \\
 \text{EQK_PI} \frac{\Gamma \vdash k_1 \equiv k_2 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash k_3 \equiv k_4 : \text{kind}}{\Gamma \vdash \Pi \alpha : k_1.k_3 \equiv \Pi \alpha : k_2.k_4 : \text{kind}} \\
 \\
 \text{EQK_SIGMA} \frac{\Gamma \vdash k_1 \equiv k_2 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash k_3 \equiv k_4 : \text{kind}}{\Gamma \vdash \Sigma \alpha : k_1.k_3 \equiv \Sigma \alpha : k_2.k_4 : \text{kind}}
 \end{array}$$

Fig. 11. Kind equivalence $\Gamma \vdash k_1 \equiv k_2 : \text{kind}$

$$\begin{array}{c}
 \text{SUBK_REFL} \frac{\Gamma \vdash k_1 \equiv k_2 : \text{kind}}{\Gamma \vdash k_1 \leq k_2 : \text{kind}} \\
 \\
 \text{SUBK_TRANS} \frac{\Gamma \vdash k_1 \leq k_2 : \text{kind} \quad \Gamma \vdash k_2 \leq k_3 : \text{kind}}{\Gamma \vdash k_1 \leq k_3 : \text{kind}} \\
 \\
 \text{SUBK_SING_T} \frac{\Gamma \vdash c : \mathbb{T}}{\Gamma \vdash S(c) \leq \mathbb{T} : \text{kind}} \\
 \\
 \text{SUBK_PI} \frac{\Gamma \vdash k'_1 \leq k_1 : \text{kind} \quad \Gamma, \alpha : k'_1 \vdash k_2 \leq k'_2 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash \Pi \alpha : k_1.k_2 \leq \Pi \alpha : k'_1.k'_2 : \text{kind}} \\
 \\
 \text{SUBK_SIGMA} \frac{\Gamma \vdash k_1 \leq k'_1 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash k_2 \leq k'_2 : \text{kind} \quad \Gamma, \alpha : k'_1 \vdash k'_2 : \text{kind}}{\Gamma \vdash \Sigma \alpha : k_1.k_2 \leq \Sigma \alpha : k'_1.k'_2 : \text{kind}}
 \end{array}$$

Fig. 12. Subkinding $\Gamma \vdash k_1 \leq k_2 : \text{kind}$

$$\begin{array}{c}
\text{OFC_VAR} \frac{\Gamma(\alpha) = k}{\Gamma \vdash \alpha : k} \quad \text{OFC_LAM} \frac{\Gamma \vdash k_1 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash c : k_2}{\Gamma \vdash \lambda \alpha : k_1. c : \Pi \alpha : k_1. k_2} \\
\\
\text{OFC_APP} \frac{\Gamma \vdash c_1 : \Pi \alpha : k_1. k_2 \quad \Gamma \vdash c_2 : k_1}{\Gamma \vdash c_1 c_2 : k_2[\alpha \mapsto c_2]} \\
\\
\text{OFC_PAIR} \frac{\Gamma \vdash c_1 : k_1 \quad \Gamma \vdash c_2 : k_2[\alpha \mapsto c_1] \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash \langle c_1, c_2 \rangle : \Sigma \alpha : k_1. k_2} \\
\\
\text{OFC_PI1} \frac{\Gamma \vdash c : \Sigma \alpha : k_1. k_2}{\Gamma \vdash \pi_1 c : k_1} \quad \text{OFC_PI2} \frac{\Gamma \vdash c : \Sigma \alpha : k_1. k_2}{\Gamma \vdash \pi_2 c : k_2[\alpha \mapsto \pi_1 c]} \\
\\
\text{OFC_STAR} \frac{}{\Gamma \vdash \star : \mathbf{1}} \quad \text{OFC_UNIT} \frac{}{\Gamma \vdash \mathbf{unit} : \mathbf{T}} \quad \text{OFC_INT} \frac{}{\Gamma \vdash \mathbf{int} : \mathbf{T}} \\
\\
\text{OFC_ARROW} \frac{\Gamma \vdash \tau_1 : \mathbf{T} \quad \Gamma \vdash \tau_2 : \mathbf{T}}{\Gamma \vdash \tau_1 \rightarrow \tau_2 : \mathbf{T}} \quad \text{OFC_PROD} \frac{\Gamma \vdash \tau_1 : \mathbf{T} \quad \Gamma \vdash \tau_2 : \mathbf{T}}{\Gamma \vdash \tau_1 \times \tau_2 : \mathbf{T}} \\
\\
\text{OFC_ALL} \frac{\Gamma \vdash k : \text{kind} \quad \Gamma, \alpha : k \vdash \tau : \mathbf{T}}{\Gamma \vdash \forall \alpha : k. \tau : \mathbf{T}} \\
\\
\text{OFC_EXISTS} \frac{\Gamma \vdash k : \text{kind} \quad \Gamma, \alpha : k \vdash \tau : \mathbf{T}}{\Gamma \vdash \exists \alpha : k. \tau : \mathbf{T}} \quad \text{OFC_SING} \frac{\Gamma \vdash c : \mathbf{T}}{\Gamma \vdash c : S(c)} \\
\\
\text{OFC_EXTPI} \frac{\Gamma \vdash c : \Pi \alpha : k_1. k_2 \quad \Gamma, \alpha : k_1 \vdash c \alpha : k'_2}{\Gamma \vdash c : \Pi \alpha : k_1. k'_2} \\
\\
\text{OFC_EXTSIGMA} \frac{\Gamma \vdash \pi_1 c : k_1 \quad \Gamma \vdash \pi_2 c : k_2[\alpha \mapsto \pi_1 c] \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash c : \Sigma \alpha : k_1. k_2} \\
\\
\text{OFC_SUBSUME} \frac{\Gamma \vdash c : k \quad \Gamma \vdash k \leq k' : \text{kind}}{\Gamma \vdash c : k'}
\end{array}$$

Fig. 13. Well-formed constructor $\Gamma \vdash c : k$

$$\begin{array}{c}
\text{EQC_REFL} \frac{\Gamma \vdash c : k}{\Gamma \vdash c \equiv c : k} \quad \text{EQC_SYMM} \frac{\Gamma \vdash c_1 \equiv c_2 : k}{\Gamma \vdash c_2 \equiv c_1 : k} \quad \text{EQC_TRANS} \frac{\Gamma \vdash c_1 \equiv c_2 : k \quad \Gamma \vdash c_2 \equiv c_3 : k}{\Gamma \vdash c_1 \equiv c_3 : k} \\
\\
\text{EQC_LAM} \frac{\Gamma \vdash k_1 \equiv k'_1 : \text{kind} \quad \Gamma, \alpha : k_1 \vdash c \equiv c' : k_2}{\Gamma \vdash \lambda \alpha : k_1. c \equiv \lambda \alpha : k'_1. c' : \Pi \alpha : k_1. k_2} \quad \text{EQC_APP} \frac{\Gamma \vdash c_1 \equiv c'_1 : \Pi \alpha : k_1. k_2 \quad \Gamma \vdash c_2 \equiv c'_2 : k_1}{\Gamma \vdash c_1 c_2 \equiv c'_1 c'_2 : k_2[\alpha \mapsto c_2]} \\
\\
\text{EQC_PAIR} \frac{\Gamma \vdash c_1 \equiv c'_1 : k_1 \quad \Gamma \vdash c_2 \equiv c'_2 : k_2[\alpha \mapsto c_1] \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash \langle c_1, c_2 \rangle \equiv \langle c'_1, c'_2 \rangle : \Sigma \alpha : k_1. k_2} \\
\\
\text{EQC_PI1} \frac{\Gamma \vdash c \equiv c' : \Sigma \alpha : k_1. k_2}{\Gamma \vdash \pi_1 c \equiv \pi_1 c' : k_1} \quad \text{EQC_PI2} \frac{\Gamma \vdash c \equiv c' : \Sigma \alpha : k_1. k_2}{\Gamma \vdash \pi_2 c \equiv \pi_2 c' : k_2[\alpha \mapsto \pi_1 c]} \\
\\
\text{EQC_ARROW} \frac{\Gamma \vdash \tau_1 \equiv \tau'_1 : \mathbb{T} \quad \Gamma \vdash \tau_2 \equiv \tau'_2 : \mathbb{T}}{\Gamma \vdash \tau_1 \rightarrow \tau_2 \equiv \tau'_1 \rightarrow \tau'_2 : \mathbb{T}} \quad \text{EQC_PROD} \frac{\Gamma \vdash \tau_1 \equiv \tau'_1 : \mathbb{T} \quad \Gamma \vdash \tau_2 \equiv \tau'_2 : \mathbb{T}}{\Gamma \vdash \tau_1 \times \tau_2 \equiv \tau'_1 \times \tau'_2 : \mathbb{T}} \\
\\
\text{EQC_ALL} \frac{\Gamma \vdash k \equiv k' : \text{kind} \quad \Gamma, \alpha : k \vdash \tau \equiv \tau' : \mathbb{T}}{\Gamma \vdash \forall \alpha : k. \tau \equiv \forall \alpha : k'. \tau' : \mathbb{T}} \\
\\
\text{EQC_EXISTS} \frac{\Gamma \vdash k \equiv k' : \text{kind} \quad \Gamma, \alpha : k \vdash \tau \equiv \tau' : \mathbb{T}}{\Gamma \vdash \exists \alpha : k. \tau \equiv \exists \alpha : k'. \tau' : \mathbb{T}} \\
\\
\text{EQC_SING} \frac{\Gamma \vdash c \equiv c' : \mathbb{T}}{\Gamma \vdash c \equiv c' : S(c)} \quad \text{EQC_SINGELIM} \frac{\Gamma \vdash c : S(c')}{\Gamma \vdash c \equiv c' : \mathbb{T}} \\
\\
\text{EQC_EXTPI} \frac{\Gamma \vdash c : \Pi \alpha : k_1. k'_2 \quad \Gamma \vdash c' : \Pi \alpha : k_1. k''_2 \quad \Gamma, \alpha : k_1 \vdash c \alpha \equiv c' \alpha : k_2}{\Gamma \vdash c \equiv c' : \Pi \alpha : k_1. k_2} \\
\\
\text{EQC_EXTPIW} \frac{\Gamma \vdash c \equiv c' : \Pi \alpha : k_1. k'_2 \quad \Gamma, \alpha : k_1 \vdash c \alpha \equiv c' \alpha : k_2}{\Gamma \vdash c \equiv c' : \Pi \alpha : k_1. k_2} \\
\\
\text{EQC_EXTSIGMA} \frac{\Gamma \vdash \pi_1 c \equiv \pi_1 c' : k_1 \quad \Gamma \vdash \pi_2 c \equiv \pi_2 c' : k_2[\alpha \mapsto \pi_1 c] \quad \Gamma, \alpha : k_1 \vdash k_2 : \text{kind}}{\Gamma \vdash c \equiv c' : \Sigma \alpha : k_1. k_2} \\
\\
\text{EQC_EXTONE} \frac{\Gamma \vdash c : 1 \quad \Gamma \vdash c' : 1}{\Gamma \vdash c \equiv c' : 1} \quad \text{EQC_SUBSUME} \frac{\Gamma \vdash c \equiv c' : k \quad \Gamma \vdash k \leq k' : \text{kind}}{\Gamma \vdash c \equiv c' : k'} \\
\\
\text{EQC_BETA} \frac{\Gamma, \alpha : k_1 \vdash c_2 : k_2 \quad \Gamma \vdash c_1 : k_1}{\Gamma \vdash (\lambda \alpha : k_1. c_2) c_1 \equiv c_2[\alpha \mapsto c_1] : k_2[\alpha \mapsto c_2]} \\
\\
\text{EQC_BETA1} \frac{\Gamma \vdash c_1 : k_1 \quad \Gamma \vdash c_2 : k_2}{\Gamma \vdash \pi_1 \langle c_1, c_2 \rangle \equiv c_1 : k_1} \quad \text{EQC_BETA2} \frac{\Gamma \vdash c_1 : k_1 \quad \Gamma \vdash c_2 : k_2}{\Gamma \vdash \pi_2 \langle c_1, c_2 \rangle \equiv c_2 : k_2}
\end{array}$$

Fig. 14. Constructor equivalence $\Gamma \vdash c_1 \equiv c_2 : k$

$$\begin{array}{c}
\text{OFT_VAR} \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau} \quad \text{OFT_STAR} \frac{}{\Gamma \vdash \star : \mathbf{unit}} \quad \text{OFT_INT} \frac{}{\Gamma \vdash n : \mathbf{int}} \\
\\
\text{OFT_LAM} \frac{\Gamma \vdash \tau_1 : \mathbf{T} \quad \Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x : \tau_1. e : \tau_1 \rightarrow \tau_2} \\
\\
\text{OFT_APP} \frac{\Gamma \vdash e_1 : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_2 : \tau_1}{\Gamma \vdash e_1 e_2 : \tau_2} \quad \text{OFT_PAIR} \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \langle e_1, e_2 \rangle : \tau_1 \times \tau_2} \\
\\
\text{OFT_PI1} \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_1 e : \tau_1} \quad \text{OFT_PI2} \frac{\Gamma \vdash e : \tau_1 \times \tau_2}{\Gamma \vdash \pi_2 e : \tau_2} \\
\\
\text{OFT_PLAM} \frac{\Gamma \vdash k : \mathbf{kind} \quad \Gamma, \alpha : k \vdash e : \tau}{\Gamma \vdash \Lambda \alpha : k. e : \forall \alpha : k. \tau} \\
\\
\text{OFT_PAPP} \frac{\Gamma \vdash e : \forall \alpha : k. \tau \quad \Gamma \vdash c : k}{\Gamma \vdash e[c] : \tau[\alpha \mapsto c]} \\
\\
\text{OFT_PACK} \frac{\Gamma \vdash c : k \quad \Gamma \vdash e : \tau[\alpha \mapsto c] \quad \Gamma, \alpha : k \vdash \tau : \mathbf{T}}{\Gamma \vdash \mathbf{pack}[c, e] \mathbf{as} \exists \alpha : k. \tau : \exists \alpha : k. \tau} \\
\\
\text{OFT_UNPACK} \frac{\Gamma \vdash e_1 : \exists \alpha : k. \tau \quad \Gamma, \alpha : k, x : \tau \vdash e_2 : \tau' \quad \Gamma \vdash \tau' : \mathbf{T}}{\Gamma \vdash \mathbf{unpack}[\alpha, x] = e_1 \mathbf{in} e_2 : \tau'} \\
\\
\text{OFT_FIX} \frac{\Gamma \vdash e : (\mathbf{unit} \rightarrow \tau) \rightarrow \tau}{\Gamma \vdash \mathbf{fix}_\tau e : \tau} \quad \text{OFT_LETT} \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \mathbf{let} x = e_1 \mathbf{in} e_2 : \tau_2} \\
\\
\text{OFT_LETM} \frac{\Gamma \vdash_1 M : \sigma \quad \Gamma, \alpha/m : \sigma \vdash e : \tau \quad \Gamma \vdash \tau : \mathbf{T}}{\Gamma \vdash \mathbf{let} \alpha/m = M \mathbf{in} e : \tau} \\
\\
\text{OFT_EXT} \frac{\Gamma \vdash_1 M : \langle \tau \rangle}{\Gamma \vdash \mathbf{Ext} M : \tau} \quad \text{OFT_EQUIV} \frac{\Gamma \vdash e : \tau \quad \Gamma \vdash \tau \equiv \tau' : \mathbf{T}}{\Gamma \vdash e : \tau'}
\end{array}$$

Fig. 15. Well-typed term $\Gamma \vdash e : \tau$

$$\begin{array}{c}
\text{OFS_ONE} \frac{}{\Gamma \vdash 1 : \text{sig}} \quad \text{OFS_STAT} \frac{\Gamma \vdash k : \text{kind}}{\Gamma \vdash \langle k \rangle : \text{sig}} \quad \text{OFS_DYN} \frac{\Gamma \vdash \tau : \mathbb{T}}{\Gamma \vdash \langle \tau \rangle : \text{sig}} \\
\\
\text{OFS_PIAPP} \frac{\Gamma \vdash \sigma_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 : \text{sig}}{\Gamma \vdash \Pi^{\text{ap}} \alpha : \alpha : \sigma_1.\sigma_2 : \text{sig}} \\
\\
\text{OFS_PIGEN} \frac{\Gamma \vdash \sigma_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 : \text{sig}}{\Gamma \vdash \Pi^{\text{gn}} \alpha : \alpha : \sigma_1.\sigma_2 : \text{sig}} \\
\\
\text{OFS_SIGMA} \frac{\Gamma \vdash \sigma_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 : \text{sig}}{\Gamma \vdash \Sigma \alpha : \sigma_1.\sigma_2 : \text{sig}}
\end{array}$$

Fig. 16. Well-formed signature $\Gamma \vdash \sigma : \text{sig}$

$$\begin{array}{c}
\text{EQS_REFL} \frac{\Gamma \vdash \sigma : \text{sig}}{\Gamma \vdash \sigma \equiv \sigma : \text{sig}} \quad \text{EQS_SYMM} \frac{\Gamma \vdash \sigma \equiv \sigma' : \text{sig}}{\Gamma \vdash \sigma' \equiv \sigma : \text{sig}} \\
\\
\text{EQS_TRANS} \frac{\Gamma \vdash \sigma \equiv \sigma'' : \text{sig} \quad \Gamma \vdash \sigma'' \equiv \sigma' : \text{sig}}{\Gamma \vdash \sigma \equiv \sigma' : \text{sig}} \quad \text{EQS_STAT} \frac{\Gamma \vdash k \equiv k' : \text{kind}}{\Gamma \vdash \langle k \rangle \equiv \langle k' \rangle : \text{sig}} \\
\\
\text{EQS_DYN} \frac{\Gamma \vdash \tau \equiv \tau' : \mathbb{T}}{\Gamma \vdash \langle \tau \rangle \equiv \langle \tau' \rangle : \text{sig}} \\
\\
\text{EQS_PIGEN} \frac{\Gamma \vdash \sigma_1 \equiv \sigma'_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 \equiv \sigma'_2 : \text{sig}}{\Gamma \vdash \Pi^{\text{gn}} \alpha : \sigma_1.\sigma_2 \equiv \Pi^{\text{gn}} \alpha : \sigma'_1.\sigma'_2 : \text{sig}} \\
\\
\text{EQS_PIAPP} \frac{\Gamma \vdash \sigma_1 \equiv \sigma'_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 \equiv \sigma'_2 : \text{sig}}{\Gamma \vdash \Pi^{\text{ap}} \alpha : \sigma_1.\sigma_2 \equiv \Pi^{\text{ap}} \alpha : \sigma'_1.\sigma'_2 : \text{sig}} \\
\\
\text{EQS_SIGMA} \frac{\Gamma \vdash \sigma_1 \equiv \sigma'_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 \equiv \sigma'_2 : \text{sig}}{\Gamma \vdash \Sigma \alpha : \sigma_1.\sigma_2 \equiv \Sigma \alpha : \sigma'_1.\sigma'_2 : \text{sig}}
\end{array}$$

Fig. 17. Signature equivalence $\Gamma \vdash \sigma \equiv \sigma' : \text{sig}$

$$\begin{array}{c}
\text{SUBS_REFL} \frac{\Gamma \vdash \sigma \equiv \sigma' : \text{sig}}{\Gamma \vdash \sigma \leq \sigma' : \text{sig}} \quad \text{SUBS_TRANS} \frac{\Gamma \vdash \sigma \leq \sigma'' : \text{sig} \quad \Gamma \vdash \sigma'' \leq \sigma' : \text{sig}}{\Gamma \vdash \sigma \leq \sigma' : \text{sig}} \\
\\
\text{SUBS_STAT} \frac{\Gamma \vdash k \leq k' : \text{kind}}{\Gamma \vdash \llbracket k \rrbracket \leq \llbracket k' \rrbracket : \text{sig}} \\
\\
\text{SUBS_PIGEN} \frac{\Gamma \vdash \sigma'_1 \leq \sigma_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma'_1) \vdash \sigma_2 \leq \sigma'_2 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 : \text{sig}}{\Gamma \vdash \Pi^{\text{gn}} \alpha : \sigma_1.\sigma_2 \leq \Pi^{\text{gn}} \alpha : \sigma'_1.\sigma'_2 : \text{sig}} \\
\\
\text{SUBS_PIAPP} \frac{\Gamma \vdash \sigma'_1 \leq \sigma_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma'_1) \vdash \sigma_2 \leq \sigma'_2 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 : \text{sig}}{\Gamma \vdash \Pi^{\text{ap}} \alpha : \sigma_1.\sigma_2 \leq \Pi^{\text{ap}} \alpha : \sigma'_1.\sigma'_2 : \text{sig}} \\
\\
\text{SUBS_SIGMA} \frac{\Gamma \vdash \sigma_1 \leq \sigma'_1 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma_1) \vdash \sigma_2 \leq \sigma'_2 : \text{sig} \quad \Gamma, \alpha : \text{Fst}(\sigma'_1) \vdash \sigma'_2 : \text{sig}}{\Gamma \vdash \Sigma \alpha : \sigma_1.\sigma_2 \leq \Sigma \alpha : \sigma'_1.\sigma'_2 : \text{sig}}
\end{array}$$

Fig. 18. Subsignature $\Gamma \vdash \sigma \leq \sigma' : \text{sig}$

$$\begin{array}{c}
\text{OFM_VAR} \frac{\Gamma(m) = \sigma}{\Gamma \vdash_P m : \sigma} \quad \text{OFM_STAR} \frac{}{\Gamma \vdash_P \star : 1} \quad \text{OFM_STAT} \frac{\Gamma \vdash c : k}{\Gamma \vdash_P \langle e \rangle : \langle k \rangle} \\
\\
\text{OFM_DYN} \frac{\Gamma \vdash e : \tau}{\Gamma \vdash_P \langle e \rangle : \langle \tau \rangle} \quad \text{OFM_LAMGN} \frac{\Gamma \vdash \sigma : \text{sig} \quad \Gamma, \alpha/m : \sigma \vdash_1 M : \sigma'}{\Gamma \vdash_P \lambda^{\text{gn}} \alpha / m : \sigma. M : \Pi^{\text{gn}} \alpha : \sigma. \sigma'} \\
\\
\text{OFM_APPGN} \frac{\Gamma \vdash_1 M_1 : \Pi^{\text{gn}} \alpha : \sigma. \sigma' \quad \Gamma \vdash_P M_2 : \sigma \quad \Gamma \vdash \text{Fst}(M_2) \gg c_2}{\Gamma \vdash_1 M_1 M_2 : \sigma'[\alpha \mapsto c_2]} \\
\\
\text{OFM_LAMAP} \frac{\Gamma \vdash \sigma : \text{sig} \quad \Gamma, \alpha/m : \sigma \vdash_P M : \sigma'}{\Gamma \vdash_P \lambda^{\text{ap}} \alpha / m : \sigma. M : \Pi^{\text{ap}} \alpha : \sigma. \sigma'} \\
\\
\text{OFM_APPAP} \frac{\Gamma \vdash_\kappa M_1 : \Pi^{\text{ap}} \alpha : \sigma. \sigma' \quad \Gamma \vdash_P M_2 : \sigma \quad \Gamma \vdash \text{Fst}(M_2) \gg c_2}{\Gamma \vdash_\kappa M_1 \cdot M_2 : \sigma'[\alpha \mapsto c_2]} \\
\\
\text{OFM_PAIR} \frac{\Gamma \vdash_\kappa M_1 : \sigma_1 \quad \Gamma \vdash_\kappa M_2 : \sigma_2 \quad \alpha \notin FV(\sigma_2)}{\Gamma \vdash_\kappa \langle M_1, M_2 \rangle : \Sigma \alpha : \sigma_1. \sigma_2} \\
\\
\text{OFM_PI1} \frac{\Gamma \vdash_P M : \Sigma \alpha : \sigma_1. \sigma_2}{\Gamma \vdash_P \pi_1 M : \sigma_1} \\
\\
\text{OFM_PI2} \frac{\Gamma \vdash_P M : \Sigma \alpha : \sigma_1. \sigma_2 \quad \Gamma \vdash \text{Fst}(M) \gg c}{\Gamma \vdash_P \pi_2 M : \sigma_2[\alpha \mapsto \pi_1 c]} \\
\\
\text{OFM_UNPACK} \frac{\Gamma \vdash e : \exists \alpha : k. \tau \quad \Gamma, \alpha : k, x : \tau \vdash_1 M : \sigma \quad \Gamma \vdash \sigma : \text{sig}}{\Gamma \vdash_1 \text{unpack}[\alpha, x] = e \text{ in } M : \sigma : \sigma} \\
\\
\text{OFM_LETT} \frac{\Gamma \vdash e : \tau \quad \Gamma, x : \tau \vdash_\kappa M : \sigma}{\Gamma \vdash_\kappa \text{let } x = e \text{ in } M : \sigma} \\
\\
\text{OFM_LETM} \frac{\Gamma \vdash_1 M_1 : \sigma \quad \Gamma, \alpha/m : \sigma \vdash_1 M_2 : \sigma' \quad \Gamma \vdash \sigma' : \text{sig}}{\Gamma \vdash_1 \text{let } \alpha/m = M_1 \text{ in } (M_2 : \sigma') : \sigma'} \\
\\
\text{OFM_SEAL} \frac{\Gamma \vdash_1 M : \sigma}{\Gamma \vdash_1 (M :> \sigma) : \sigma} \\
\\
\text{OFM_EXTSTAT} \frac{\Gamma \vdash_P M : \langle k' \rangle \quad \Gamma \vdash \text{Fst}(M) \gg c \quad \Gamma \vdash c : k}{\Gamma \vdash_P M : \langle k \rangle} \\
\\
\text{OFM_EXTPI} \frac{\Gamma \vdash_P M : \Pi^{\text{ap}} \alpha : \sigma_1. \sigma_2' \quad \Gamma, \alpha/m : \sigma_1 \vdash_P M \cdot m : \sigma_2}{\Gamma \vdash_P M : \Pi^{\text{ap}} \alpha : \sigma_1. \sigma_2} \\
\\
\text{OFM_EXTSIGMA} \frac{\Gamma \vdash_P \pi_1 M : \sigma_1 \quad \Gamma \vdash_P \pi_2 M : \sigma_2 \quad \alpha \notin FV(\sigma_2)}{\Gamma \vdash_P M : \Sigma \alpha : \sigma_1. \sigma_2} \\
\\
\text{OFM_FORGET} \frac{\Gamma \vdash_P M : \sigma}{\Gamma \vdash_1 M : \sigma} \quad \text{OFM_SUBSUME} \frac{\Gamma \vdash_\kappa M : \sigma \quad \Gamma \vdash \sigma \leq \sigma' : \text{sig}}{\Gamma \vdash_\kappa M : \sigma'}
\end{array}$$

Fig. 19. Well-formed module $\Gamma \vdash_\kappa M : \sigma$

$$\begin{array}{c}
\text{OFC_VAR} \frac{}{\alpha : S(\mathbf{int}) \vdash \alpha : S(\mathbf{int})} \\
\text{EQC_SINGELIM} \frac{}{\alpha : S(\mathbf{int}) \vdash \alpha \equiv \mathbf{int} : \mathbf{T}} \\
\text{EQC_SYMM} \frac{}{\alpha : S(\mathbf{int}) \vdash \mathbf{int} \equiv \alpha : \mathbf{T}} \\
\text{OFC_INT} \frac{}{\vdash \mathbf{int} : \mathbf{T}} \\
\text{SUBK_SING_T} \frac{}{\vdash S(\mathbf{int}) \leq \mathbf{T} : \text{kind}} \\
\text{SUBS_STAT} \frac{}{\vdash \langle S(\mathbf{int}) \rangle \leq \langle \mathbf{T} \rangle : \text{sig}} \\
\text{SUBS_SIGMA} \frac{}{\vdash \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle \leq \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle} \\
\text{EQS_DYN} \frac{}{\alpha : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \equiv \langle \alpha \rangle : \text{sig}} \\
\text{SUBS_REFL} \frac{}{\alpha : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \leq \langle \alpha \rangle : \text{sig}} \\
\text{OFS_DYN} \frac{}{\alpha : \mathbf{T} \vdash \langle \alpha \rangle : \text{sig}}
\end{array}$$

Fig. 20. Derivation of $\vdash \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle \leq \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle$

Example 13 (Opaque signature). We consider a module $M = \langle \langle \mathbf{int} \rangle, \langle 0 \rangle \rangle$ and show that it has the opaque signature $\sigma_O = \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle$ ¹³. We then have that M is a pure module of the signature σ_O (i.e. $\vdash_P M : \sigma_O$).

First, we have that M is a module of a transparent signature, i.e. $\vdash_P M : \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle$ by instantiating the `ofm_pair` rule.

$$\text{OFM_PAIR} \frac{\vdash_P \langle \mathbf{int} \rangle : \langle S(\mathbf{int}) \rangle \quad \vdash_P \langle 0 \rangle : \langle \mathbf{int} \rangle \quad \alpha \notin FV(\langle \mathbf{int} \rangle)}{\vdash_P M : \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle}$$

Next, we have that $\vdash \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle \leq \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle$, by the derivation described in Fig. 20:

Finally, it follows that $\vdash_P M : \sigma_O$.

$$\text{OFM_SUBSUME} \frac{\vdash_P \langle \langle \mathbf{int} \rangle, \langle 0 \rangle \rangle : \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle \quad \vdash \Sigma \alpha : \langle S(\mathbf{int}) \rangle . \langle \mathbf{int} \rangle \leq \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle}{\vdash_P M : \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle}$$

From [14], we have the following lemma about the correctness of $\mathbf{Fst}(M) \gg c$ operation.

Lemma 6. *If $\Gamma \vdash_P M : \sigma$ then $\Gamma \vdash \mathbf{Fst}(M) \gg c$ and $\Gamma \vdash c : \mathbf{Fst}(\sigma)$.*

To facilitate the proofs about TRNI for ML, from the static semantics, we have the following lemma.

Lemma 7 (Weakening). *Suppose that $\vdash \Gamma, \alpha : k$ ok. It follows that:*

- if $\Gamma \vdash \sigma : \text{sig}$, then $\Gamma, \alpha : k \vdash \sigma : \text{sig}$,
- if $\Gamma \vdash c : k'$, then $\Gamma, \alpha : k \vdash c : k'$,
- if $\Gamma \vdash k' : \text{kind}$, then $\Gamma, \alpha : k \vdash k' : \text{kind}$,
- if $\Gamma \vdash k_1 \equiv k_2 : \text{kind}$, then $\Gamma, \alpha : k \vdash k_1 \equiv k_2 : \text{kind}$,

¹³ Notice that in Example 12, we have $M = \langle \langle \mathbf{int} \rangle, \langle \langle 0 \rangle, \langle \lambda x : \mathbf{int}. e \rangle \rangle \rangle$ and $\sigma_O = \Sigma \alpha : \langle \mathbf{T} \rangle . \Sigma \beta : \langle \alpha \rangle . \langle \alpha \rightarrow \tau_f \rangle$. From the static semantics, we can also derive that $\vdash_P M : \sigma_O$. Here, to simplify the presentation, we have $M = \langle \langle \mathbf{int} \rangle, \langle 0 \rangle \rangle$ and $\sigma_O = \Sigma \alpha : \langle \mathbf{T} \rangle . \langle \alpha \rangle$

- if $\Gamma \vdash k_1 \leq k_2 : \text{kind}$, then $\Gamma, \alpha : k \vdash k_1 \leq k_2 : \text{kind}$,
- if $\Gamma \vdash c_1 \equiv c_2 : k'$, then $\Gamma, \alpha : k \vdash c_1 \equiv c_2 : k'$.

Proof. We prove this lemma by induction on the derivation of $\Gamma \vdash \sigma : \text{sig}$, $\Gamma \vdash c : k'$, $\Gamma \vdash k' : \text{kind}$, $\Gamma \vdash k_1 \equiv k_2 : \text{kind}$, $\Gamma \vdash k_1 \leq k_2 : \text{kind}$, and $\Gamma \vdash c_1 \equiv c_2 : k'$.

Dynamic semantics. The dynamic semantics is given by call-by value semantics. We have dynamic semantics for terms $\Gamma \vdash e \rightarrow e'$ and for modules $\Gamma \vdash M \rightarrow M'$ (see Fig. 21 and Fig. 22), where the context Γ is only used to extract the static part of module values. Open term values and module values are as below.

$$\begin{array}{ll}
 v := x \mid \star \mid n \mid \lambda x : \tau. e \mid \langle v, v \rangle \mid \Lambda \alpha : k. e & \text{Term values} \\
 \mid \text{pack}[c, v] \text{ as } \exists \alpha : k. \tau & \\
 V := m \mid \star \mid \langle c \rangle \mid \langle v \rangle \mid \langle V, V \rangle & \text{Module values} \\
 \mid \lambda^{\text{gn}} \alpha / m : \sigma. M \mid \lambda^{\text{ap}} \alpha / m : \sigma. M &
 \end{array}$$

In the `tstep_fix` rule, $\lambda_- : \mathbf{unit}. \text{fix}_\tau e$ means that the term variable bound by λ is a fresh variable. To be precise, the variable must not be in $\text{dom}(\Gamma)$, and in addition it should be canonically chosen, to maintain strict determinacy of evaluation. In Crary's deBruin representation this is automatic.

We use V, W as metavariables for module values. We write $e \downarrow$ when the evaluation of e terminates. Similarly, we have $M \downarrow$.

$$\begin{array}{c}
\text{TSTEP_APP1} \frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash e_1 \ e_2 \rightarrow e'_1 \ e_2} \qquad \text{TSTEP_APP2} \frac{\Gamma \vdash e_2 \rightarrow e'_2}{\Gamma \vdash v_1 \ e_2 \rightarrow v_1 \ e'_2} \\
\\
\text{TSTEP_APP3} \frac{}{\Gamma \vdash (\lambda x : \tau. e_1) \ v_2 \rightarrow e_1[x \mapsto v_2]} \\
\\
\text{TSTEP_PAIR1} \frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \langle e_1, e_2 \rangle \rightarrow \langle e'_1, e_2 \rangle} \qquad \text{TSTEP_PAIR2} \frac{\Gamma \vdash e_2 \rightarrow e'_2}{\Gamma \vdash \langle v_1, e_2 \rangle \rightarrow \langle v_1, e'_2 \rangle} \\
\\
\text{TSTEP_PI11} \frac{\Gamma \vdash e \rightarrow e'}{\Gamma \vdash \pi_1 e \rightarrow \pi_1 e'} \qquad \text{TSTEP_PI12} \frac{}{\Gamma \vdash \pi_1 \langle v_1, v_2 \rangle \rightarrow v_1} \\
\\
\text{TSTEP_PI21} \frac{\Gamma \vdash e \rightarrow e'}{\Gamma \vdash \pi_2 e \rightarrow \pi_2 e'} \qquad \text{TSTEP_PI22} \frac{}{\Gamma \vdash \pi_2 \langle v_1, v_2 \rangle \rightarrow v_2} \\
\\
\text{TSTEP_PAPP1} \frac{\Gamma \vdash e \rightarrow e'}{\Gamma \vdash e[c] \rightarrow e'[c]} \qquad \text{TSTEP_PAPP2} \frac{}{\Gamma \vdash (\lambda \alpha : k. e)[c] \rightarrow e[\alpha \mapsto c]} \\
\\
\text{TSTEP_PACK} \frac{\Gamma \vdash e \rightarrow e'}{\Gamma \vdash \text{pack}[c, e] \text{ as } \exists \alpha : k. \tau \rightarrow \text{pack}[c, e'] \text{ as } \exists \alpha : k. \tau} \\
\\
\text{TSTEP_UNPACK1} \frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{unpack}[\alpha, x] = e_1 \text{ in } e_2 \rightarrow \text{unpack}[\alpha, x] = e'_1 \text{ in } e_2} \\
\\
\text{TSTEP_UNPACK2} \frac{}{\Gamma \vdash \text{unpack}[\alpha, x] = (\text{pack}[c, v] \text{ as } \exists \alpha : k. \tau) \text{ in } e_2 \rightarrow e[\alpha \mapsto c, x \mapsto v]} \\
\\
\text{TSTEP_FIX} \frac{}{\Gamma \vdash \text{fix}_\tau e \rightarrow e \ (\lambda_- : \mathbf{unit}. \text{fix}_\tau e)} \\
\\
\text{TSTEP_LETT1} \frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{let } x = e_1 \text{ in } e_2 \rightarrow \text{let } x = e'_1 \text{ in } e_2} \\
\\
\text{TSTEP_LETT2} \frac{}{\Gamma \vdash \text{let } x = v_1 \text{ in } e_2 \rightarrow e_2[x \mapsto v_1]} \\
\\
\text{TSTEP_LETM1} \frac{\Gamma \vdash M \rightarrow M'}{\Gamma \vdash \text{let } \alpha/m = M \text{ in } e \rightarrow \text{let } \alpha/m = M' \text{ in } e} \\
\\
\text{TSTEP_LETM2} \frac{\Gamma \vdash \mathbf{Fst}(V) \gg c}{\Gamma \vdash \text{let } \alpha/m = V \text{ in } e \rightarrow e[\alpha \mapsto c, m \mapsto V]} \\
\\
\text{TSTEP_EXT1} \frac{\Gamma \vdash M \rightarrow M'}{\Gamma \vdash \mathbf{Ext} \ M \rightarrow \mathbf{Ext} \ M'} \qquad \text{TSTEP_EXT2} \frac{}{\Gamma \vdash \mathbf{Ext} \ \langle v \rangle \rightarrow v}
\end{array}$$

Fig. 21. Dynamic semantics - Terms $\Gamma \vdash e \rightarrow e'$

$$\begin{array}{c}
\text{MSTEP_DYN} \frac{\Gamma \vdash e \rightarrow e'}{\Gamma \vdash \langle e \rangle \rightarrow \langle e' \rangle} \quad \text{MSTEP_APPGN1} \frac{\Gamma \vdash M_1 \rightarrow M'_1}{\Gamma \vdash M_1 \ M_2 \rightarrow M'_1 \ M_2} \\
\text{MSTEP_APPGN2} \frac{\Gamma \vdash M_2 \rightarrow M'_2}{\Gamma \vdash V_1 \ M_2 \rightarrow V_1 \ M'_2} \\
\text{MSTEP_APPGN3} \frac{\Gamma \vdash \text{Fst}(V_2) \gg c_2}{\Gamma \vdash (\lambda^{\text{gn}} \alpha / m : \sigma.M_1) \ V_2 \rightarrow M_1[\alpha \mapsto c_2, m \mapsto V_2]} \\
\text{MSTEP_APPAP1} \frac{\Gamma \vdash M_1 \rightarrow M'_1}{\Gamma \vdash M_1 \cdot M_2 \rightarrow M'_1 \cdot M_2} \quad \text{MSTEP_APPAP2} \frac{\Gamma \vdash M_2 \rightarrow M'_2}{\Gamma \vdash V_1 \cdot M_2 \rightarrow V_1 \cdot M'_2} \\
\text{MSTEP_APPAP3} \frac{\Gamma \vdash \text{Fst}(V_2) \gg c_2}{\Gamma \vdash (\lambda^{\text{ap}} \alpha / m : \sigma.M_1) \ V_2 \rightarrow M_2[\alpha \mapsto c_2, m \mapsto V_2]} \\
\text{MSTEP_PAIR1} \frac{\Gamma \vdash M_1 \rightarrow M'_1}{\Gamma \vdash \langle M_1, M_2 \rangle \rightarrow \langle M'_1, M_2 \rangle} \quad \text{MSTEP_PAIR2} \frac{\Gamma \vdash M_2 \rightarrow M'_2}{\Gamma \vdash \langle V_1, M_2 \rangle \rightarrow \langle V_1, M'_2 \rangle} \\
\text{MSTEP_PI11} \frac{\Gamma \vdash M \rightarrow M'}{\Gamma \vdash \pi_1 M \rightarrow \pi_1 M'} \quad \text{MSTEP_PI12} \frac{}{\Gamma \vdash \pi_1 \langle V_1, V_2 \rangle \rightarrow V_1} \\
\text{MSTEP_PI21} \frac{\Gamma \vdash M \rightarrow M'}{\Gamma \vdash \pi_2 M \rightarrow \pi_2 M'} \quad \text{MSTEP_PI22} \frac{}{\Gamma \vdash \pi_2 \langle V_1, V_2 \rangle \rightarrow V_2} \\
\text{MSTEP_UNPACK1} \frac{\Gamma \vdash e \rightarrow e'}{\Gamma \vdash \text{unpack}[\alpha, x] = e \text{ in } (M : \sigma) \rightarrow \text{unpack}[\alpha, x] = e' \text{ in } (M : \sigma)} \\
\text{MSTEP_UNPACK2} \frac{}{\Gamma \vdash \text{unpack}[\alpha, x] = (\text{pack}[c, v] \text{ as } \exists \alpha : k.\tau \text{ in } (M : \sigma) \rightarrow M[\alpha \mapsto c, x \mapsto v])} \\
\text{MSTEP_LETT1} \frac{\Gamma \vdash e_1 \rightarrow e'_1}{\Gamma \vdash \text{let } x = e_1 \text{ in } M_2 \rightarrow \text{let } x = e'_1 \text{ in } M_2} \\
\text{MSTEP_LETT2} \frac{}{\Gamma \vdash \text{let } x = v_1 \text{ in } M_2 \rightarrow M_2[x \mapsto v_1]} \\
\text{MSTEP_LETM1} \frac{\Gamma \vdash M_1 \rightarrow M'_1}{\Gamma \vdash \text{let } \alpha / m = M_1 \text{ in } (M_2 : \sigma) \rightarrow \text{let } \alpha / m = M'_1 \text{ in } (M_2 : \sigma)} \\
\text{MSTEP_LETM2} \frac{\Gamma \vdash \text{Fst}(V) \gg c}{\Gamma \vdash \text{let } \alpha / m = V \text{ in } (M : \sigma) \rightarrow M[\alpha \mapsto c, m \mapsto V]} \\
\text{MSTEP_SEAL} \frac{}{\Gamma \vdash (M :> \sigma) \rightarrow M}
\end{array}$$

Fig. 22. Dynamic semantics - Modules $\Gamma \vdash M \rightarrow M'$

F.2 Logical relation

In order to define logical relation, we define some auxiliary notions as in [14]. Given a relation $R \in \text{Rel}(\tau, \tau')$, R^s contains continuations that agree on values related by R . Conversely, given a relation S on continuations, related continuations by S agree on terms in S^t . From $_^s$ and $_^t$, we define Pitts closed relations.

Definition 4 (Closure). For $R \in \text{Rel}(\tau, \tau')$, define

$$R^s \triangleq \{\langle v : \tau \rightarrow \mathbf{unit}, v' : \tau \rightarrow \mathbf{unit} \rangle \mid \forall \langle w, w' \rangle \in R : vw \Downarrow \Leftrightarrow v'w' \Downarrow\}.$$

For $S \in \text{Rel}(\tau \rightarrow \mathbf{unit}, \tau' \rightarrow \mathbf{unit})$, define $S^t \triangleq \{\langle w : \tau, w' : \tau' \rangle \mid \forall \langle v, v' \rangle \in S, vw \Downarrow \Leftrightarrow v'w' \Downarrow\}.$

For $R \in \text{Rel}(\tau, \tau')$, define

$$R^{\text{ev}} \triangleq \{\langle e : \tau, e' : \tau' \rangle \mid e \Downarrow \Leftrightarrow e' \Downarrow, \forall v, v'. e \rightarrow^* v \implies e' \rightarrow^* v' \implies \langle v, v' \rangle \in R\}$$

Relation $R \in \text{Rel}(\tau, \tau')$ is Pitts closed if $R = R^{\text{st}}$.

We have similar definitions for relations defined on closed signatures σ_1 and σ_2 , where the signatures for continuations are $\Pi^{\text{gn}}\alpha : \sigma_1.1$ and $\Pi^{\text{gn}}\alpha : \sigma_2.1$, which can be abbreviated as $\sigma_1 \rightarrow 1$ and $\sigma_2 \rightarrow 1$.

Apropos the **stev** closure, we may be able to infer indirectly that two terms are related by R^{stev} when these two terms depend on terms related by Q^{stev} .

Definition 5. Suppose $x : \varrho \vdash e : \tau$. We say that x is active in e if for all closed e' s.t. $\vdash e' : \varrho$, $e[x \mapsto e'] \Downarrow$ implies $e' \Downarrow$.

Lemma 8. Suppose $Q \in \text{Rel}(\varrho_1, \varrho_2)$, $R \in \text{Rel}(\tau_1, \tau_2)$, $x : \varrho_i \vdash e_i : \tau_i$, and x is active in e_1 and e_2 . If for all $\langle v_1, v_2 \rangle \in Q$, $\langle e_1[x \mapsto v_1], e_2[x \mapsto v_2] \rangle \in R^{\text{stev}}$, then

$$\forall \langle e'_1, e'_2 \rangle \in Q^{\text{stev}}. \langle e_1[x \mapsto e'_1], e_2[x \mapsto e'_2] \rangle \in R^{\text{stev}}.$$

From [14] and Lemma `wf_mr` in [15], we have the following lemma about properties of logical interpretations of constructors, kinds and signatures.

Lemma 9. Suppose that $\vdash \Gamma \text{ ok}$ and $\langle \rho, \rho' \rangle \in \llbracket \Gamma \rrbracket$. Then

- If $\Gamma \vdash c : k$, then $\langle \rho_L(c), \rho_R(c), \llbracket c \rrbracket_\rho, \llbracket c \rrbracket_{\rho'} \rangle \in \llbracket k \rrbracket_\rho$.
- If $\Gamma \vdash k_1 \equiv k_2$, then $\llbracket k_1 \rrbracket_\rho = \llbracket k_2 \rrbracket_\rho$.
- If $\Gamma \vdash \sigma : \text{sig}$, then $\llbracket \sigma \rrbracket_\rho = \llbracket \sigma \rrbracket_{\rho'}$.
- If $\Gamma \vdash \sigma \equiv \sigma' : \text{sig}$, then $\llbracket \sigma \rrbracket_\rho = \llbracket \sigma' \rrbracket_\rho$.

Precandidate. As in [14], we next present simple kinds which are used in the definitions of logical interpretations of dependent kinds. A *simple kind* is a kind that does not have singleton kinds. Given a kind k , $\text{simp}(k)$ returns a simple kind by replacing singleton kinds in k with \top . A *candidate* of a kind is a pre-candidate which is in an interpretation of a kind. We use Q as a meta-variable for pre-candidates in general, Φ for pre-candidates over function kinds, P for pre-candidates over pair kinds, and R for pre-candidates over \top .

$$\begin{aligned}
\text{Val} &\triangleq \{v \mid \exists \tau. \vdash v : \tau\} \\
\text{Con} &\triangleq \{c \mid \exists k. \vdash c : k\} \\
\text{PreCand}_\top &\triangleq \mathcal{P}(\text{Val} \times \text{Val}) \\
\text{PreCand}_1 &\triangleq \{\langle \rangle\} \\
\text{PreCand}_{k_1 \rightarrow k_2} &\triangleq \text{Con} \times \text{Con} \times \text{PreCand}_{k_1} \rightarrow \text{PreCand}_{k_2} \\
\text{PreCand}_{k_1 \times k_2} &\triangleq \text{PreCand}_{k_1} \times \text{PreCand}_{k_2}
\end{aligned}$$

Logical interpretations for kinds and constructors. Following [14], we generalize ρ presented in §2 to a mapping that maps constructor variables to tuples of the form $\langle c, c', Q \rangle$, term variables to tuples of the form $\langle v, v' \rangle$, and module variables to tuples of the form $\langle V, V' \rangle$. Notice that in the simple language, for any $\alpha \in \text{dom}(\rho)$, $\rho(\alpha) = R \in \text{Rel}(\tau_1, \tau_2)$ for some τ_1 and τ_2 . If we use the notation in this section, then we have that $\rho(\alpha) = \langle \tau_1, \tau_2, R \rangle$. We write ρ_L and ρ_R for the substitutions that map every variable in the domain of ρ to respectively the first element and the second element of the tuple that ρ maps that variable to. If we do not have module variables, then, $\rho_L(_)$ is similar to $\delta_1 \gamma_1(_)$ and $\rho_R(_)$ is similar to $\delta_2 \gamma_2(_)$ in §2, where δ_1 and δ_2 are type substitutions in ρ , and γ_1 and γ_2 are term substitutions in ρ .

Logical interpretations for kinds and constructors are presented in Fig. 23¹⁴. The interpretation of a kind k is a tuple $\langle c, c', Q, Q' \rangle$ where c and c' are closed constructors of the kind k , and Q and Q' are candidates relating c and c' . Notice that since pre-candidates are defined only for simple kinds, in the logical interpretations of $\Pi\alpha : k_1.k_2$ and $\Sigma\alpha : k_1.k_2$, we use respectively $\text{PreCand}_{\text{simp}(\Pi\alpha:k_1.k_2)}$ and $\text{PreCand}_{\text{simp}(\Sigma\alpha:k_1.k_2)}$. In addition, in the definition for $\Pi\alpha : k_1.k_2$, since Φ and Φ' are pre-candidates of $\text{simp}(\Pi\alpha : k_1.k_2)$, we require that Q and Q' are pre-candidates of $\text{simp}(k_1)$ ¹⁵. The definitions for types of \top (e.g. $\tau_1 \rightarrow \tau_2$) are similar to the ones of the simple language.

Logical interpretations for signatures. Logical interpretations for signatures are presented in Fig. 24. The logical interpretation of a signature σ of modules is a set of $\langle V_1, V_2, Q \rangle$ where the dynamic values in V_1 and V_2 are related at their types, and the static part (constructors) are related by Q . For example, we consider the signature $\Sigma\alpha : \langle k \rangle. \langle \tau \rangle$. The logical interpretation of this signature with a ρ is a set of $\langle \langle V_1, V'_1 \rangle, \langle V_2, V'_2 \rangle, \langle P, P' \rangle \rangle$ where (1) $V_1 = \langle c \rangle$, $V_2 = \langle c' \rangle$ for some c and c' s.t. $\langle c, c', P, P' \rangle \in \llbracket k \rrbracket_\eta$; and (2) $V_2 = \langle v \rangle$, $V'_2 = \langle v' \rangle$ for some v and v' s.t. $\langle v, v' \rangle \in \llbracket \tau \rrbracket_\rho$ (from the definition of $\llbracket \langle \tau \rangle \rrbracket_\rho$ we also know that $P' = \langle \rangle$).

¹⁴ The presentation of logical interpretations here is similar to the one in [14]. Notice that we can write definitions of logical interpretations in the form of inference rules as in §2.

¹⁵ Notice that from definition of $\text{PreCand}_{\Pi\alpha:k_1.k_2}$, we have that $\Phi\langle d, d', Q \rangle$ and $\Phi\langle d'', d''', Q' \rangle$ are in $\text{PreCand}_{\text{simp}(k_2)}$.

$$\begin{aligned}
\llbracket \mathbf{T} \rrbracket_\rho &\triangleq \{ \langle \tau, \tau', R, R \rangle \mid \vdash \tau, \tau' : \mathbf{T}, R \in \text{Rel}(\tau, \tau'), R \text{ Pitts closed} \} \\
\llbracket S(c) \rrbracket_\rho &\triangleq \{ \langle \tau, \tau, \llbracket c \rrbracket_\rho, \llbracket c \rrbracket_\rho \rangle \mid \vdash \tau \equiv \rho_L(c) : \mathbf{T}, \vdash \tau' \equiv \rho_R(c) : \mathbf{T}, \llbracket c \rrbracket_\rho \in \text{Rel}(\tau, \tau'), \llbracket c \rrbracket_\rho \text{ Pitts closed} \} \\
\llbracket \Pi\alpha : k_1.k_2 \rrbracket_\rho &\triangleq \{ \langle c, c', \Phi, \Phi' \rangle \mid \vdash c : \rho_L(\Pi\alpha : k_1.k_2), \vdash c' : \rho_R(\Pi\alpha : k_1.k_2), \\
&\quad \Phi, \Phi' \in \text{PreCand}_{\text{simp}(\Pi\alpha : k_1.k_2)}, \\
&\quad \forall d, d', Q, Q', d'', d'''. Q, Q' \in \text{PreCand}_{\text{simp}(k_1)} \implies \\
&\quad \langle d, d', Q, Q' \rangle \in \llbracket k_1 \rrbracket_\rho \implies \\
&\quad \vdash d \equiv d'' : \rho_L(k_1) \implies \vdash d' \equiv d''' : \rho_R(k_1) \implies \\
&\quad \langle cd, c'd', \Phi(d, d', Q), \Phi'(d'', d''', Q') \rangle \in \llbracket k_2 \rrbracket_{\rho, \alpha \mapsto \langle d, d', Q \rangle} \} \\
\llbracket \Sigma\alpha : k_1.k_2 \rrbracket_\rho &\triangleq \{ \langle c, c', P, P' \rangle \mid \vdash c : \rho_L(\Sigma\alpha : k_1.k_2), \vdash c' : \rho_R(\Sigma\alpha : k_1.k_2), \\
&\quad P, P' \in \text{PreCand}_{\text{simp}(\Sigma\alpha : k_1.k_2)} \\
&\quad \langle \pi_1 c, \pi_1 c', \pi_1 P, \pi_1 P' \rangle \in \llbracket k_1 \rrbracket_\rho, \\
&\quad \langle \pi_2 c, \pi_2 c', \pi_2 P, \pi_2 P' \rangle \in \llbracket k_2 \rrbracket_{\rho, \alpha \mapsto \langle \pi_1 c, \pi_1 c', \pi_1 P \rangle} \} \\
\llbracket \mathbf{1} \rrbracket_\rho &\triangleq \{ \langle c, c', \langle \rangle, \langle \rangle \rangle \mid \vdash c : \mathbf{1}, \vdash c' : \mathbf{1} \} \\
\llbracket k \rrbracket_\rho^{\text{set}} &\triangleq \{ \langle c, c', Q \rangle \mid \langle c, c', Q, Q \rangle \in \llbracket k \rrbracket_\rho \} \\
\llbracket \alpha \rrbracket_\rho &\triangleq Q, \text{ where } \rho(\alpha) = \langle c, c', Q \rangle & \llbracket \lambda\alpha : k.c \rrbracket_\rho &\triangleq \lambda\langle d, d', Q \rangle \in \llbracket k \rrbracket_\rho^{\text{set}}. \llbracket c \rrbracket_{\rho, \alpha \mapsto \langle d, d', Q \rangle} \\
\llbracket c_1 c_2 \rrbracket_\rho &\triangleq \llbracket c_1 \rrbracket_\rho \langle \rho_L(c_2), \rho_R(c_2), \llbracket c_2 \rrbracket_\rho \rangle & \llbracket \langle c_1, c_2 \rangle \rrbracket_\rho &\triangleq \langle \llbracket c_1 \rrbracket_\rho, \llbracket c_2 \rrbracket_\rho \rangle \\
\llbracket \pi_i c \rrbracket_\rho &\triangleq \pi_i \llbracket c \rrbracket_\rho & \llbracket \star \rrbracket_\rho &\triangleq \langle \rangle \\
\llbracket \text{unit} \rrbracket_\rho &\triangleq \{ \langle \star, \star \rangle \} \\
\llbracket \text{int} \rrbracket_\rho &\triangleq \{ \langle v, v \rangle \mid \vdash v : \text{int} \} \\
\llbracket \tau_1 \rightarrow \tau_2 \rrbracket_\rho &\triangleq \{ \langle v_1, v_2 \rangle \mid \vdash v_1 : \rho_L(\tau_1 \rightarrow \tau_2), \vdash v_2 : \rho_R(\tau_1 \rightarrow \tau_2), \forall \langle v'_1, v'_2 \rangle \in \llbracket \tau_1 \rrbracket_\rho. \langle v_1 v'_1, v_2 v'_2 \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}} \} \\
\llbracket \tau_1 \times \tau_2 \rrbracket_\rho &\triangleq \{ \langle \langle v_1, v_2 \rangle, \langle v'_1, v'_2 \rangle \rangle \mid \vdash \langle v_1, v_2 \rangle : \rho_L(\tau_1 \times \tau_2), \vdash \langle v'_1, v'_2 \rangle : \rho_R(\tau_1 \times \tau_2), \\
&\quad \langle v_1, v'_1 \rangle \in \llbracket \tau_1 \rrbracket_\rho, \langle v_2, v'_2 \rangle \in \llbracket \tau_2 \rrbracket_\rho \} \\
\llbracket \forall\alpha : k.\tau \rrbracket_\rho &\triangleq \{ \langle v, v' \rangle \mid \vdash v : \rho_L(\forall\alpha : k.\tau), \vdash v' : \rho_R(\forall\alpha : k.\tau), \\
&\quad \forall \langle c, c', Q \rangle \in \llbracket k \rrbracket_\rho^{\text{set}}. \langle v[c], v'[c'] \rangle \in \llbracket \tau \rrbracket_{\rho[\alpha \mapsto \langle c, c', Q \rangle]} \} \\
\llbracket \exists\alpha : k.\tau \rrbracket_\rho &\triangleq \{ \langle v, v' \rangle \mid \vdash v : \rho_L(\exists\alpha : k.\tau), \vdash v' : \rho_R(\exists\alpha : k.\tau), \\
&\quad \exists \langle c, c', Q \rangle \in \llbracket k \rrbracket_\rho^{\text{set}}, \exists v_0, v'_0, k', k'', \tau', \tau''. \\
&\quad v = \text{pack}[c, v_0] \text{ as } \exists\alpha : k'. \tau', v' = \text{pack}[c', v'_0] \text{ as } \exists\alpha : k''. \tau'', \\
&\quad \langle v_0, v'_0 \rangle \in \llbracket \tau \rrbracket_{\rho, \alpha \mapsto \langle c, c', Q \rangle}^{\text{st}} \}
\end{aligned}$$

Fig. 23. Logical interpretation (kinds and constructors)

$$\begin{aligned}
\llbracket 1 \rrbracket_\rho &\triangleq \{\langle \star, \star, \langle \rangle \rangle\} \\
\llbracket \langle k \rangle \rrbracket_\rho &\triangleq \{\langle \langle c \rangle, \langle c' \rangle, Q \rangle \mid \langle c, c', Q \rangle \in \llbracket k \rrbracket_\rho^{\text{set}}\} \\
\llbracket \langle \tau \rangle \rrbracket_\rho &\triangleq \{\langle \langle v \rangle, \langle v' \rangle, \langle \rangle \rangle \mid \vdash v : \rho_L(\tau), \vdash v : \rho_R(\tau), \langle v, v' \rangle \in \llbracket \tau \rrbracket_\rho\} \\
\llbracket \Pi^{\text{gn}} \alpha : \sigma_1. \sigma_2 \rrbracket_\rho &\triangleq \{\langle V, V', \langle \rangle \rangle \mid \vdash V : \rho_L(\Pi^{\text{gn}} \alpha : \sigma_1. \sigma_2), \vdash V' : \rho_R(\Pi^{\text{gn}} \alpha : \sigma_1. \sigma_2), \\
&\quad \forall \langle W, W', Q \rangle \in \llbracket \sigma_1 \rrbracket_\rho. \langle VW, V'W' \rangle \in \llbracket \sigma_2 \rrbracket_{\rho, \alpha \mapsto \langle \text{Fst}(W), \text{Fst}(W'), Q \rangle}^{\text{iev}}\} \\
\llbracket \Pi^{\text{ap}} \alpha : \sigma_1. \sigma_2 \rrbracket_\rho &\triangleq \{\langle V, V', \Phi \rangle \mid \vdash V : \rho_L(\Pi^{\text{ap}} \alpha : \sigma_1. \sigma_2), \vdash V' : \rho_R(\Pi^{\text{ap}} \alpha : \sigma_1. \sigma_2), \\
&\quad \langle \text{Fst}(V), \text{Fst}(V'), \Phi \rangle \in \llbracket \text{Fst}(\Pi^{\text{ap}} \alpha : \sigma_1. \sigma_2) \rrbracket_\rho^{\text{set}}, \\
&\quad \forall \langle W, W', Q \rangle \in \llbracket \sigma_1 \rrbracket_\rho. \langle V \cdot W, V' \cdot W' \rangle \in \llbracket \sigma_2 \rrbracket_{\rho, \alpha \mapsto \langle \text{Fst}(W), \text{Fst}(W'), Q \rangle}^{\text{pev}}\} \\
\llbracket \Sigma \alpha : \sigma_1. \sigma_2 \rrbracket_\rho &\triangleq \{\langle V, V', P \rangle \mid \vdash V : \rho_L(\Sigma \alpha : \sigma_1. \sigma_2), \vdash V' : \rho_R(\Sigma \alpha : \sigma_1. \sigma_2), \\
&\quad \exists V_1, V'_1, V_2, V'_2. V = \langle V_1, V_2 \rangle, V' = \langle V'_1, V'_2 \rangle, \langle V_1, V'_1, \pi_1 P \rangle \in \llbracket \sigma_1 \rrbracket_\rho, \\
&\quad \langle V_2, V'_2, \pi_2 P \rangle \in \llbracket \sigma_2 \rrbracket_{\rho, \alpha \mapsto \langle \text{Fst}(V_1), \text{Fst}(V'_1), \pi_1 P \rangle}\} \\
\llbracket \sigma \rrbracket_\rho^{\text{pev}} &\triangleq \{\langle M, M', Q \rangle \mid \vdash_P M : \rho_L(\sigma), \vdash_P M' : \rho_R(\sigma), M \Downarrow \Leftrightarrow M' \Downarrow, \\
&\quad \forall V, V'. M \rightarrow^* V \implies M' \rightarrow^* V' \implies \langle V, V', Q \rangle \in \llbracket \sigma \rrbracket_\rho\} \\
\llbracket \sigma \rrbracket_\rho^{\text{i}} &\triangleq \{\langle V, V' \rangle \mid \exists Q. \langle V, V', Q \rangle \in \llbracket \sigma \rrbracket_\rho\}^{\text{st}}
\end{aligned}$$

Fig. 24. Logical interpretation (signatures)

Definition 6. We say that $\langle \rho, \rho' \rangle \in \llbracket I \rrbracket$ if whenever $\Gamma(\alpha) = k$, there exists $\rho(\alpha) = \langle c_1, c_2, Q \rangle$ and $\rho'(\alpha) = \langle c'_1, c'_2, Q' \rangle$ s.t. $\vdash c_1 \equiv c'_1 : \rho_L(k)$, $\vdash c_2 \equiv c'_2 : \rho_R(k)$, and $\langle c_1, c_2, Q, Q' \rangle \in \llbracket k \rrbracket_\rho$,

We say that $\rho \in \llbracket I \rrbracket^{\text{full}}$ if $\langle \rho, \rho \rangle \in \llbracket I \rrbracket$ and

- for all $x : \tau \in \Gamma$, there exists $\rho(x) = \langle v_1, v_2 \rangle$ s.t. $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_\rho$,
- for all $\alpha/m : \sigma \in \Gamma$, there exists $\rho(m) = \langle V_1, V_2 \rangle$ and $\rho(\alpha) = \langle \text{Fst}(V_1), \text{Fst}(V_2), Q \rangle$ s.t. $\langle V_1, V_2, Q \rangle \in \llbracket \sigma \rrbracket_\rho$.

Terms e and e' are *logically equivalent* at τ in Γ (written as $\Gamma \vdash e \sim e' : \tau$) if $\vdash \Gamma \text{ ok}$ implies $\Gamma \vdash e, e' : \tau$, and for all $\rho \in \llbracket \Gamma \rrbracket^{\text{full}}$, $\langle \rho_L(e), \rho_R(e') \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}$. Notice that equivalence holds vacuously, if Γ is not well formed, but we are never interested in such Γ .

Theorem 6 (Abstraction theorem). Suppose that $\vdash \Gamma \text{ ok}$. If $\Gamma \vdash e : \tau$, then $\Gamma \vdash e \sim e : \tau$.

In [14], there are similar results for pure modules and impure modules. Later we express security in terms of sealed modules, but our security proof only relies on the abstraction theorem for expressions.

G TRNI for the Module Calculus

This section recapitulates the development of §4 but using an encoding suited to the module calculus. The free theorem that typing implies security (Theorem 4)

$$\begin{aligned}
\langle\langle L \rangle\rangle_C &\triangleq \begin{cases} 1 & \text{if } L = [], \\ \Sigma\alpha_x : \langle S(\mathbf{int}) \rangle. \Sigma\alpha : \langle \mathbf{int} \rangle. \langle\langle L' \rangle\rangle_C & \text{if } L = x :: L', x \notin \text{dom}(\mathbf{F}_{\mathcal{P}}), \\ \Sigma\alpha_f : \langle S(\mathbf{int}) \rangle. \Sigma\alpha_1 : \langle \mathbf{int} \rangle. \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau_f \rangle. \langle\langle L' \rangle\rangle_C & \text{if } L = x :: L', \mathbf{F}_{\mathcal{P}}(x) = f, \end{cases} \\
\langle\langle L \rangle\rangle_P &\triangleq \begin{cases} 1 & \text{if } L = [], \\ \Sigma\alpha_x : \langle \mathbf{T} \rangle. \Sigma\alpha : \langle \alpha_x \rangle. \langle\langle L' \rangle\rangle_P & \text{if } L = x :: L', x \notin \text{dom}(\mathbf{F}_{\mathcal{P}}), \\ \Sigma\alpha_f : \langle \mathbf{T} \rangle. \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau_f \rangle. \langle\langle L' \rangle\rangle_P & \text{if } L = x :: L', \mathbf{F}_{\mathcal{P}}(x) = f, \end{cases}
\end{aligned}$$

Fig. 25. Transparent and opaque signatures for a policy \mathcal{P} .

is formulated for an open term in context of the public view, as in Theorem 2. We then develop a “wrapper” to encapsulate the typing problem in a closed form. That could facilitate use of an unmodified ML compiler without recourse to an API for the typechecker.

G.1 Declassification policy encoding

In this section, we present the encoding for declassification policies by using the module calculus. Here, a declassification function can be written in the module calculus with recursive functions. However, for simplicity and for coherent policy, we assume—as in §3—that the applications of declassifiers on confidential input values always terminate. In §4, a view is a typing context that declares variables for inputs and for declassifiers. Here, those are gathered in a signature and the view is a context that declares a module of that signature.

Let $L \subseteq \mathbf{V}_{\mathcal{P}}$ be a finite list of distinct confidential input variables from $\mathbf{V}_{\mathcal{P}}$. An empty list is $[]$. We write $x :: L$ to concatenate a confidential input variable to L . In §4 we define operations $\langle\langle - \rangle\rangle_C$ and $\langle\langle - \rangle\rangle_P$ that apply to policy variables and declassifiers, yielding the encoding of policy as typing contexts. Here we use the same notation, but apply the operations to variable lists and encode policy as signatures. First, we define $\langle\langle L \rangle\rangle_C$ to return a transparent signature of the policy. It is defined inductively as described in Fig. 25. As in §4, we use fresh constructor variables with names that indicate their role in the encoding. For a confidential input x , basically, the signature is a pair containing information about the kind of its type, its type, and the types of associated declassifiers. For example, for x that can be declassified via f , the signature contains: (1) the kind of the type of x : $\langle S(\mathbf{int}) \rangle$, (2) the type of x : $\langle \mathbf{int} \rangle$, and (3) the type of f : $\langle \mathbf{int} \rightarrow \tau_f \rangle$.

Example 14 (Transparent signature). For \mathcal{P}_{OE} (Example 1), since $f = \lambda x : \mathbf{int}. x \bmod 2$ is of the type $\mathbf{int} \rightarrow \mathbf{int}$, by applying the third case in $\langle\langle - \rangle\rangle_C$ with $\tau_f = \mathbf{int}$, we get the signature

$$\langle\langle x \rangle\rangle_C = \Sigma\alpha_f : \langle S(\mathbf{int}) \rangle. \Sigma\alpha_1 : \langle \mathbf{int} \rangle. \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \mathbf{int} \rangle. 1$$

that can be abbreviated as $\langle\langle S(\mathbf{int}) \rangle, \langle\langle \mathbf{int} \rangle, \langle\langle \mathbf{int} \rightarrow \mathbf{int} \rangle, 1 \rangle \rangle$.

In ML it looks like

```
sig type t=int val x:int val f:int->int end.
```

We overload $\langle\langle L \rangle\rangle_P$ to get an opaque signature of the policy. The idea is similar to $\langle\langle L \rangle\rangle_C$, except that here we use constructor variables of the T kind for types of confidential inputs and in types of declassification functions. The definition is described in Fig. 25.

Example 15 (Opaque signature). For \mathcal{P}_{OE} (Example 1), since $f = \lambda x : \mathbf{int}. x \bmod 2$ is of the type $\mathbf{int} \rightarrow \mathbf{int}$, by applying the third case in $\langle\langle - \rangle\rangle_P$ with $\tau_f = \mathbf{int}$, we get the signature

$$\langle\langle x \rangle\rangle_P = \Sigma \alpha_f : \langle\mathsf{T}\rangle. \Sigma \alpha_1 : \langle\alpha_f\rangle. \Sigma \alpha_2 : \langle\alpha_f \rightarrow \mathbf{int}\rangle. 1$$

that can be abbreviated as $\Sigma \alpha_f : \langle\mathsf{T}\rangle. \langle\langle \alpha_f \rangle\rangle. \langle\langle \alpha_f \rightarrow \mathbf{int} \rangle\rangle. 1$. In ML it looks like `sig type t val x:t val f: t->int end.`

Hereafter, we abuse $\mathbf{V}_{\mathcal{P}}$ and use it as a list and we write $\sigma_{\mathcal{P}}^C$ and $\sigma_{\mathcal{P}}$ to mean respectively $\langle\langle \mathbf{V}_{\mathcal{P}} \rangle\rangle_C$ and $\langle\langle \mathbf{V}_{\mathcal{P}} \rangle\rangle_P$.

In order to define TRNI, we define the confidential view and the public view as in §4. The confidential view is based on the constructed transparent signature $\sigma_{\mathcal{P}}^C$, and the public view is based on the constructed opaque signature $\sigma_{\mathcal{P}}$.

$$\Gamma_C^{\mathcal{P}} \triangleq \alpha_{\mathcal{P}} / m_{\mathcal{P}} : \sigma_{\mathcal{P}}^C \quad \Gamma_P^{\mathcal{P}} \triangleq \alpha_{\mathcal{P}} / m_{\mathcal{P}} : \sigma_{\mathcal{P}} \quad (2)$$

To express what in Example 2 and Example 3 (for \mathcal{P}_{OE}) is written $x_f x$, in the module calculus x is accessed as $\mathbf{Ext}(\pi_1(\pi_2 m_{\mathcal{P}_{OE}}))$ and x_f is accessed as $\mathbf{Ext}(\pi_1(\pi_2(\pi_2 m_{\mathcal{P}_{OE}})))$.

From the definitions, we have that $\sigma_{\mathcal{P}}^C$ and $\sigma_{\mathcal{P}}$ are closed and well-formed signatures, and $\sigma_{\mathcal{P}}^C$ is a subsignature of $\sigma_{\mathcal{P}}$.

G.2 TRNI

In order to define an environment ρ for the policy, we define relations R_x and R_f similar to the relations $\mathcal{I}_V[\alpha_x]$ and $\mathcal{I}_V[\alpha_f]$ in §4.

$$\begin{aligned} R_x &= \{ \langle v_1, v_2 \rangle \mid \vdash v_1 : \mathbf{int}, \vdash v_2 : \mathbf{int} \} \\ R_f &= \{ \langle v_1, v_2 \rangle \mid \vdash v_1 : \mathbf{int}, \vdash v_2 : \mathbf{int}, \langle f \ v_1, f \ v_2 \rangle \in \llbracket \tau_f \rrbracket_{\emptyset}^{\text{ev}} \} \end{aligned}$$

Notice that \emptyset in $\llbracket \tau \rrbracket_{\emptyset}^{\text{ev}}$ is the empty environment.

Given a list L of confidential inputs from \mathcal{P} and an environment ρ , we say that $\rho \in |L|_{\mathcal{P}}$ when

- ρ maps $m_{\mathcal{P}}$ in $\Gamma_P^{\mathcal{P}}$ to related module values V_1 and V_2 for some V_1 and V_2 s.t. V_1 and V_2 are of the transparent signature $\langle\langle \mathbf{V}_{\mathcal{P}} \rangle\rangle_C$ and functions in V_1 and V_2 are declassification functions from the policy, and the confidential values in V_1 and V_2 are related by R_x , R_f , or $R_{f \circ a}$ according to the policy, and

- ρ maps $\alpha_{\mathcal{P}}$ to a tuple $\langle c_1, c_2, Q \rangle$ where c_1 and c_2 are static parts from respectively V_1 and V_2 , and Q depends on the policy. That is if an element in Q is corresponding to an $\alpha_x : \langle \mathbb{T} \rangle$, then this element is R_x , if an element in Q is corresponding to an $\alpha_f : \langle \mathbb{T} \rangle$, then this element is R_f , and if an element in Q is corresponding to an $\alpha_{f \circ a} : \langle \mathbb{T} \rangle$, then this element is $R_{f \circ a}$.

The definition of $\rho \in |L|_{\mathcal{P}}$ is as below. Hereafter, we write $\rho \models^{\text{full}} \mathcal{P}$ when $\rho \in |\mathbf{V}_{\mathcal{P}}|_{\mathcal{P}}$.

Definition 7 (full environments for \mathcal{P}). Given $L \subseteq \mathbf{V}_{\mathcal{P}}$, we define the set $|L|_{\mathcal{P}}$ of environments by $\rho \in |L|_{\mathcal{P}}$ iff $\text{dom}(\rho) = \{\alpha_{\mathcal{P}}, m_{\mathcal{P}}\}$ and

- if $L = []$ then $\rho(m_{\mathcal{P}}) = \langle \star, \star \rangle$ and $\rho(\alpha_{\mathcal{P}}) = \langle \star, \star, \langle \rangle \rangle$,
- if $L = x :: L'$ and $x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$ then there are $\langle v_1, v_2 \rangle \in R_x$ and $\rho' \in |L'|_{\mathcal{P}}$ with

$$\begin{aligned} \rho(m_{\mathcal{P}}) &= \langle \langle \langle \mathbf{int} \rangle, \langle \langle v_1 \rangle, V'_1 \rangle \rangle, \langle \langle \mathbf{int} \rangle, \langle \langle v_2 \rangle, V'_2 \rangle \rangle \rangle, \\ \rho(\alpha_{\mathcal{P}}) &= \langle \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle, \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle, \langle R_x, \langle \langle \rangle, Q' \rangle \rangle \rangle, \end{aligned}$$

- where $\rho'(m_{\mathcal{P}}) = \langle V'_1, V'_2 \rangle$ and $\rho'(\alpha_{\mathcal{P}}) = \langle c'_1, c'_2, Q' \rangle$,
- if $L = x :: L'$ and $\mathbf{F}_{\mathcal{P}}(x) = f$ then there are $\langle v_1, v_2 \rangle \in R_f$ and $\rho' \in |L'|_{\mathcal{P}}$ with

$$\begin{aligned} \rho(m_{\mathcal{P}}) &= \langle \langle \langle \mathbf{int} \rangle, \langle \langle v_1 \rangle, \langle \langle f \rangle, V'_1 \rangle \rangle \rangle, \\ &\quad \langle \langle \mathbf{int} \rangle, \langle \langle v_2 \rangle, \langle \langle f \rangle, V'_2 \rangle \rangle \rangle \rangle, \\ \rho(\alpha_{\mathcal{P}}) &= \langle \langle \mathbf{int}, \langle \star, \langle \star, c'_1 \rangle \rangle \rangle, \langle \mathbf{int}, \langle \star, \langle \star, c'_2 \rangle \rangle \rangle, \\ &\quad \langle R_f, \langle \langle \rangle, \langle \langle \rangle, Q' \rangle \rangle \rangle \rangle, \end{aligned}$$

where $\rho'(m_{\mathcal{P}}) = \langle V'_1, V'_2 \rangle$ and $\rho'(\alpha_{\mathcal{P}}) = \langle c'_1, c'_2, Q' \rangle$.

Example 16 ($\rho \models^{\text{full}} \mathcal{P}_{OE}$). In this example, we present a full environment for \mathcal{P}_{OE} . We first define R_f , where $f = \lambda x : \mathbf{int}. x \bmod 2$.

$$R_f = \{ \langle v_1, v_2 \rangle \mid \vdash v_1 : \mathbf{int}, \vdash v_2 : \mathbf{int}, (v_1 \bmod 2) =_{\mathbf{int}} (v_2 \bmod 2) \}.$$

Following the definition of $\rho \models^{\text{full}} \mathcal{P}_{OE}$, we construct ρ as below. Notice that $\langle 2, 4 \rangle \in R_f$.

$$\begin{aligned} \rho(m_{\mathcal{P}_{OE}}) &= \langle \langle \langle \mathbf{int} \rangle, \langle \langle 2 \rangle, \langle \langle \lambda x : \mathbf{int}. x \bmod 2 \rangle, \star \rangle \rangle \rangle, \\ &\quad \langle \langle \mathbf{int} \rangle, \langle \langle 4 \rangle, \langle \langle \lambda x : \mathbf{int}. x \bmod 2 \rangle, \star \rangle \rangle \rangle \rangle, \\ \rho(\alpha_{\mathcal{P}_{OE}}) &= \langle \langle \mathbf{int}, \langle \star, \langle \star, \star \rangle \rangle \rangle, \\ &\quad \langle \mathbf{int}, \langle \star, \langle \star, \star \rangle \rangle \rangle, \langle R_f, \langle \langle \rangle, \langle \langle \rangle, \langle \rangle \rangle \rangle \rangle \rangle. \end{aligned}$$

It follows that $\rho \models^{\text{full}} \mathcal{P}_{OE}$.

Next we prove that if τ is a type in the public view $\Gamma_{\mathcal{P}}^{\mathcal{P}}$, then all its logical interpretations are the same.

Lemma 10. *If $\rho_1 \models^{\text{full}} \mathcal{P}$, $\rho_2 \models^{\text{full}} \mathcal{P}$, and $\Gamma_P^{\mathcal{P}} \vdash \tau : \mathbb{T}$, then $\llbracket \tau \rrbracket_{\rho_1} = \llbracket \tau \rrbracket_{\rho_2}$.*

Therefore, we define indistinguishability based on an arbitrary $\rho \models^{\text{full}} \mathcal{P}$.

Definition 8 (Indistinguishability). *Suppose ρ and τ satisfy $\rho \models^{\text{full}} \mathcal{P}$ and $\Gamma_P^{\mathcal{P}} \vdash \tau : \mathbb{T}$.*

- *Values v_1 and v_2 are indistinguishable at τ (written as $\langle v_1, v_2 \rangle \in \mathcal{I}_V \llbracket \tau \rrbracket$) if $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_{\rho}$.*
- *Terms e_1 and e_2 are indistinguishable at τ (written as $\langle e_1, e_2 \rangle \in \mathcal{I}_E \llbracket \tau \rrbracket$) if $\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_{\rho}^{\text{ev}}$.*

Example 17 (Indistinguishability). We consider \mathcal{P}_{OE} (Example 1). As described in Example 15, the opaque signature of the policy is $\sigma_{\mathcal{P}_{OE}} = \Sigma \alpha_f : \langle \mathbb{T} \rangle. \Sigma \alpha_1 : \langle \alpha_f \rangle. \Sigma \alpha_2 : \langle \alpha_f \rightarrow \mathbf{int} \rangle. 1$. Thus, the public view $\Gamma_P^{\mathcal{P}_{OE}}$ is $\alpha_{\mathcal{P}_{OE}} / m_{\mathcal{P}_{OE}} : \sigma_{\mathcal{P}_{OE}}$. Notice that since $\alpha_{\mathcal{P}_{OE}}$ and $m_{\mathcal{P}_{OE}}$ are twinned, it follows that $\alpha_{\mathcal{P}_{OE}}$ is of the kind $\text{Fst}(\sigma_{\mathcal{P}}) = \Sigma \alpha_f : \mathbb{T}. \Sigma \alpha_1 : 1. \Sigma \alpha_2 : 1.1$.

We consider the type $\pi_1 \alpha_{\mathcal{P}_{OE}}$. By a rule for well-formed constructors (rule `ofc_pi1`), we have that $\Gamma_P^{\mathcal{P}_{OE}} \vdash \pi_1 \alpha_{\mathcal{P}_{OE}} : \mathbb{T}$. Thus, we can define indistinguishability for this type. As presented in Example 16, $\rho \models^{\text{full}} \mathcal{P}_{OE}$. Therefore, we have that $\mathcal{I}_V \llbracket \pi_1 \alpha_{\mathcal{P}_{OE}} \rrbracket = \llbracket \pi_1 \alpha_{\mathcal{P}_{OE}} \rrbracket_{\rho} = R_f$ (the definition of R_f is in Example 16).

Next, we define TRNI for the module calculus. The definition here is similar to the one in §4.

Definition 9 (TRNI for the module calculus). *A term e is $\text{TRNI}(\mathcal{P}, \tau)$ if $\Gamma_C^{\mathcal{P}} \vdash e$, and $\Gamma_P^{\mathcal{P}} \vdash \tau : \mathbb{T}$, and for all $\rho \models^{\text{full}} \mathcal{P}$, it follows that $\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_E \llbracket \tau \rrbracket$.*

Example 18. We consider the program $e = (\text{Ext}(\pi_1(\pi_2(\pi_2 m_{\mathcal{P}_{OE}})))) (\text{Ext}(\pi_1(\pi_2 m_{\mathcal{P}_{OE}})))$, which is corresponding to the program x_f in Example 6, as noted following Eqn. (2). We now check e with the definition of TRNI. We consider an arbitrary $\rho \models^{\text{full}} \mathcal{P}_{OE}$. As described in Example 16, ρ is as below, where $\langle v_1, v_2 \rangle \in R_f$.

$$\begin{aligned} \rho(m_{\mathcal{P}_{OE}}) &= \langle \langle \langle \mathbf{int} \rangle, \langle \langle v_1 \rangle, \langle \langle \lambda x : \mathbf{int}. x \text{ mod } 2 \rangle, \star \rangle \rangle \rangle, \\ &\quad \langle \langle \mathbf{int} \rangle, \langle \langle v_2 \rangle, \langle \langle \lambda x : \mathbf{int}. x \text{ mod } 2 \rangle, \star \rangle \rangle \rangle \rangle, \\ \rho(\alpha_{\mathcal{P}_{OE}}) &= \langle \langle \mathbf{int}, \langle \star, \langle \star, \star \rangle \rangle \rangle, \langle \mathbf{int}, \langle \star, \langle \star, \star \rangle \rangle \rangle, \langle R_f, \langle \langle \rangle, \langle \langle \rangle, \langle \rangle \rangle \rangle \rangle \rangle. \end{aligned}$$

We have that $\rho_L(e) = f \ v_1 = v_1 \text{ mod } 2$ and $\rho_R(e) = f \ v_2 = v_2 \text{ mod } 2$. Since $\langle v_1, v_2 \rangle \in R_f$, we have that $(v_1 \text{ mod } 2) =_{\mathbf{int}} (v_2 \text{ mod } 2)$. Thus, $\langle \rho_L(e), \rho_R(e) \rangle \in \llbracket \mathbf{int} \rrbracket_{\rho}^{\text{ev}}$. In other words, $\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_E \llbracket \mathbf{int} \rrbracket$. Therefore, e is $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$.

G.3 Free theorem: typing in the public view implies security

To apply the abstraction theorem to get the free theorem, we need the following.

Lemma 11. *Suppose that $\rho \models^{\text{full}} \mathcal{P}$. It follows that $\rho \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket^{\text{full}}$.*

Lemma 12. *If $\Gamma_P^{\mathcal{P}} \vdash e : \tau$, then $\Gamma_C^{\mathcal{P}} \vdash e$.*

Theorem 4. *If $\Gamma_P^{\mathcal{P}} \vdash e : \tau$, then e is $\text{TRNI}(\mathcal{P}, \tau)$.*

Proof. Since $\Gamma_P^{\mathcal{P}} \vdash e : \tau$, from Theorem 6, we have that $\Gamma_P^{\mathcal{P}} \vdash e \sim e : \tau$. Thus, for any $\rho \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket^{\text{full}}$, it follows that:

$$\langle \rho_L(e), \rho_R(e) \rangle \in \llbracket \tau \rrbracket_{\rho}^{\text{ev}}.$$

We consider an arbitrary ρ s.t. $\rho \models^{\text{full}} \mathcal{P}$. From Lemma 11, it follows that $\rho \in \llbracket \Gamma_P^{\mathcal{P}} \rrbracket^{\text{full}}$. As proven above, we have that $\langle \rho_L(e), \rho_R(e) \rangle \in \llbracket \tau \rrbracket_{\rho}^{\text{ev}}$. From the definition of indistinguishability, we have that $\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_E \llbracket \tau \rrbracket$. In addition, since $\Gamma_P^{\mathcal{P}} \vdash e : \tau$, from Lemma 12, it follows that $\Gamma_C^{\mathcal{P}} \vdash e$. Therefore, e is $\text{TRNI}(\mathcal{P}, \tau)$.

Example 19 (Typing implies TRNI). We consider the policy \mathcal{P}_{OE} . As described in Example 14 and Example 15, the transparent signature and the opaque signature of the policy are as below.

$$\begin{aligned} \sigma_{\mathcal{P}_{OE}}^C &= \Sigma \alpha_f : \langle S(\mathbf{int}) \rangle. \Sigma \alpha_1 : \langle \mathbf{int} \rangle. \Sigma \alpha_2 : \langle \mathbf{int} \rightarrow \mathbf{int} \rangle. 1 \\ \sigma_{\mathcal{P}_{OE}} &= \Sigma \alpha_f : \langle T \rangle. \Sigma \alpha_1 : \langle \alpha_f \rangle. \Sigma \alpha_2 : \langle \alpha_f \rightarrow \mathbf{int} \rangle. 1 \end{aligned}$$

Thus, the confidential view is $\Gamma_C^{\mathcal{P}_{OE}} = \alpha_{\mathcal{P}_{OE}} / m_{\mathcal{P}_{OE}} : \sigma_{\mathcal{P}_{OE}}^C$, and the public view is $\Gamma_P^{\mathcal{P}_{OE}} = \alpha_{\mathcal{P}_{OE}} / m_{\mathcal{P}_{OE}} : \sigma_{\mathcal{P}_{OE}}$. We now look at the program $e = e_1 e_2$, where $e_1 = (\text{Ext}(\pi_1(\pi_2(m_{\mathcal{P}_{OE}}))))$ and $e_2 = (\text{Ext}(\pi_1(\pi_2(m_{\mathcal{P}_{OE}}))))$. This program is corresponding to the program $x_f x$ in Example 6, as noted following definition (2). We have that $\Gamma_C^{\mathcal{P}_{OE}} \vdash e_1 : \mathbf{int} \rightarrow \mathbf{int}$, $\Gamma_C^{\mathcal{P}_{OE}} \vdash e_2 : \mathbf{int}$, $\Gamma_P^{\mathcal{P}_{OE}} \vdash e_1 : \pi_1 \alpha_{\mathcal{P}_{OE}} \rightarrow \mathbf{int}$, and $\Gamma_P^{\mathcal{P}_{OE}} \vdash e_2 : \pi_1 \alpha_{\mathcal{P}_{OE}}$. Therefore, we have that $\Gamma_C^{\mathcal{P}_{OE}} \vdash e : \mathbf{int}$, and $\Gamma_P^{\mathcal{P}_{OE}} \vdash e : \mathbf{int}$ and hence, from Theorem 4, the program is $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$.

Example 20. The purpose of this example is similar to the one of Example 7: to illustrate that if a program is well-typed in the confidential view and is not $\text{TRNI}(\mathcal{P}, \tau)$ for some τ well-formed in the public view, then the type of the program in the public view is not equivalent to τ or the program is not well-typed in the public view.

We consider the policy \mathcal{P}_{OE} and the program $\text{Ext}(\pi_1(\pi_2(m_{\mathcal{P}_{OE}})))$, which is corresponding to the program x in Example 7. This program is not $\text{TRNI}(\mathcal{P}_{OE}, \mathbf{int})$ since $\text{Ext}(\pi_1(\pi_2(m_{\mathcal{P}_{OE}})))$ itself is confidential and cannot be directly declassified. In the public view of the policy, the type of this program is $\pi_1 \alpha_{\mathcal{P}_{OE}}$ which is not equivalent to \mathbf{int} .

We consider another program: $e = (\text{Ext}(\pi_1(\pi_2(m_{\mathcal{P}_{OE}})))) \text{ mod } 3$, which is corresponding to the program $x \text{ mod } 3$ in Example 7. This program is not $\text{TRNI}(\mathcal{P}_{OE}, \pi_1 \alpha_{\mathcal{P}_{OE}})$ since it may map indistinguishable inputs to non-indistinguishable outputs. For example, we consider $\rho \models^{\text{full}} \mathcal{P}_{OE}$ presented in Example 17. We have that $\rho_L(e) = 2 \text{ mod } 3 = 2$, and $\rho_R(e) = 4 \text{ mod } 3 = 1$. As described in Example 17, $\mathcal{I}_V \llbracket \pi_1 \alpha_{\mathcal{P}_{OE}} \rrbracket = R_f = \{ \langle v_1, v_2 \rangle \mid (v_1 \text{ mod } 2) =_{\mathbf{int}} (v_2 \text{ mod } 2) \}$. Therefore $\langle 1, 2 \rangle \notin \mathcal{I}_V \llbracket \pi_1 \alpha_{\mathcal{P}_{OE}} \rrbracket$.

As explained above, e is not $\text{TRNI}(\mathcal{P}_{OE}, \pi_1 \alpha_{\mathcal{P}_{OE}})$. In the public view, it is not well-typed since the type of $\text{Ext}(\pi_1(\pi_2(m_{\mathcal{P}_{OE}})))$ is $\pi_1 \alpha_{\mathcal{P}_{OE}}$, which is not equivalent to \mathbf{int} , and mod expects \mathbf{int} arguments.

G.4 Wrapper

In this section, we will transform an open term to a closed module. We then prove that if the closed module is well-typed in the empty context, then the original open term is well-typed in the public view and hence, e is TRNI. Thus we can use our approach with ordinary ML implementations.

If the source programs are already parameterized by one module for their confidential inputs and their declassification functions, then there is no need to modify source programs at all.

We next define a wrapper that wraps e with the information from the public view.

$$\text{wrap}_{\mathcal{P}}(e) \triangleq \lambda^{\text{gn}} \alpha_{\mathcal{P}}, m_{\mathcal{P}} : \sigma_{\mathcal{P}}. \langle e \rangle$$

From the construction, we have that if $\text{wrap}_{\mathcal{P}}(e)$ is well-typed in the empty context, then the original term is also well-typed in the public view. In addition, we can infer the type of the original term in the public view. These results yield, by Theorem 4, that the original term is TRNI when the wrapper is well-typed.

Theorem 7. *If $\vdash_{\mathcal{P}} \text{wrap}_{\mathcal{P}}(e) : \Pi^{\text{gn}} \alpha_{\mathcal{P}} : \sigma_{\mathcal{P}}. \langle \tau \rangle$, then e is $\text{TRNI}(\mathcal{P}, \tau)$.*

Example 21. In this example, we combine the ideas presented in §A.2, §G.1, and §G.4 to encode a complex policy which is inspired by two-factor authentication. The policy \mathcal{P}_{Aut} involves two confidential passwords and two declassifiers `checking1` and `checking2` written in SML as below, where `input1` and `input2` are respectively the first input and the second input from a user. Notice that `checking2` takes a tuple of two passwords as its input.

```
fun checking1(password1:int) =
  if (password1 = input1) then 1 else 0
fun checking2(passwords:int*int) =
  if ((#1 passwords) = input1) then
    if ((#2 passwords) = input2) then 1 else 0
  else 2
```

Using the ideas presented in §A.2, we introduce a new variable which corresponding to the tuple of two passwords. The confidential and public signatures of \mathcal{P}_{Aut} in the module calculus are as below, where f_1 and f_2 are corresponding to `checking1` and `checking2`.

$$\begin{aligned} \sigma_{\mathcal{P}_{\text{Aut}}}^C &= \Sigma \alpha_{f_1} : \langle S(\text{int}) \rangle. \Sigma \alpha_1 : \langle \text{int} \rangle. \Sigma \alpha_2 : \langle \text{int} \rightarrow \text{int} \rangle. \\ &\quad \Sigma \alpha_{x_2} : \langle S(\text{int}) \rangle. \Sigma \alpha_3 : \langle \text{int} \rangle. \\ &\quad \Sigma \alpha_{f_2} : \langle S(\text{int} \times \text{int}) \rangle. \Sigma \alpha_4 : \langle \text{int} \times \text{int} \rangle. \\ &\quad \Sigma \alpha_5 : \langle \text{int} \times \text{int} \rightarrow \text{int} \rangle. 1 \\ \sigma_{\mathcal{P}_{\text{Aut}}} &= \Sigma \alpha_{f_1} : \langle \top \rangle. \Sigma \alpha_1 : \langle \alpha_{f_1} \rangle. \Sigma \alpha_2 : \langle \alpha_{f_1} \rightarrow \text{int} \rangle. \\ &\quad \Sigma \alpha_{x_2} : \langle \top \rangle. \Sigma \alpha_3 : \langle \alpha_{x_2} \rangle. \\ &\quad \Sigma \alpha_{f_2} : \langle \top \rangle. \Sigma \alpha_4 : \langle \alpha_{f_2} \rangle. \\ &\quad \Sigma \alpha_5 : \langle \alpha_{f_2} \rightarrow \text{int} \rangle. 1 \end{aligned}$$

The confidential and public signatures of \mathcal{P}_{Aut} in SML are as below.

```
signature traSIG=sig
  type t1 = int
  val password1:t1
  val checking1:t1->int
  type t2 = int
  val password2:t2
  type t3 = int * int
  val passwords:t3
  val checking2:t3 ->int
end

signature opaSIG=sig
  type t1
  val password1:t1
  val checking1:t1->int
  type t2
  val password2:t2
  type t3
  val passwords:t3
  val checking2:t3->int
end
```

As in §A.2, we require that for $\rho \models^{\text{full}} \mathcal{P}_{\text{Aut}}$, ρ_L and ρ_R are consistent. That is when $\rho_L(m_{\mathcal{P}_{\text{Aut}}}) = V_L$, then $V_L.\text{passwords} = \langle V_L.\text{password1}, V_L.\text{password2} \rangle$ (and we have a similar requirement for ρ_R). In order to define indistinguishability for \mathcal{P}_{Aut} , we need to define $\rho \models^{\text{full}} \mathcal{P}_{\text{Aut}}$,¹⁶ and hence, we define R_{f_1} , R_{x_2} , R_{f_2} as below.

$$\begin{aligned}
R_{f_1} &= \{ \langle v_1, v'_1 \rangle \mid \vdash v_1, v'_1 : \mathbf{int}, \\
&\quad v_1 = v'_1 = \mathbf{input1} \vee \\
&\quad (v_1 \neq \mathbf{input1} \wedge v'_1 \neq \mathbf{input1}) \} \\
R_{x_2} &= \{ \langle v_1, v'_1 \rangle \mid \vdash v_2, v'_2 : \mathbf{int} \} \\
R_{f_2} &= \{ \langle \langle v_1, v_2 \rangle, \langle v'_1, v'_2 \rangle \rangle \mid \vdash v_1, v'_1, v_2, v'_2 : \mathbf{int} \\
&\quad (v_1 = v'_1 = \mathbf{input1} \wedge v_2 = v'_2 = \mathbf{input2}) \vee \\
&\quad (v_1 = v'_1 = \mathbf{input1} \wedge v_2 \neq \mathbf{input2} \wedge \\
&\quad \quad v'_2 \neq \mathbf{input2}) \vee \\
&\quad (v_1 \neq \mathbf{input1} \wedge v'_1 \neq \mathbf{input1}) \}
\end{aligned}$$

By using the wrapper presented above, we can check that programs

$$m_{\mathcal{P}_{\text{Aut}}}.\text{checking2} \quad m_{\mathcal{P}_{\text{Aut}}}.\text{passwords},$$

and $m_{\mathcal{P}_{\text{Aut}}}.\text{checking1} \quad m_{\mathcal{P}_{\text{Aut}}}.\text{password1}$ are $\text{TRNI}(\mathcal{P}_{\text{Aut}}, \mathbf{int})$, where $m_{\mathcal{P}_{\text{Aut}}}$ is the module variable in confidential and public views of \mathcal{P}_{Aut} .

¹⁶ Notice that as noted in the main text, ρ_L and ρ_R are consistent.

Remark 3 (On wrapper). We may choose an applicative functor for wrapping the original program. However, w.r.t. this choice, we need to handle more cases in proofs. Thus, we choose a generative functor.

H Proofs for TRNI for the Module Calculus

H.1 Properties of the encoding

Lemma 13. *For any $L \subseteq \mathbf{V}_{\mathcal{P}}$, it follows that $\vdash \langle\langle L \rangle\rangle_C : \text{sig}$.*

Proof. We prove the lemma by induction on L . We have four cases.

Case 1: $L = \square$. We have that $\langle\langle \mathbf{V}_{\mathcal{P}} \rangle\rangle_C = 1$. From the ofs_one rule, it follows that $\vdash 1 : \text{sig}$.

Case 2: $L = x :: L', x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$. We have that $\langle\langle L \rangle\rangle_C = \Sigma \alpha_x : \langle\langle S(\mathbf{int}) \rangle\rangle . \Sigma \alpha : \langle\langle \mathbf{int} \rangle\rangle . \langle\langle L' \rangle\rangle_C$.

From IH, we have that $\vdash \langle\langle L' \rangle\rangle_C : \text{sig}$. Thus, we have the following derivation. Notice that $\text{Fst}(\langle\langle c \rangle\rangle) = 1$ for any c and $\text{Fst}(\langle\langle S(\mathbf{int}) \rangle\rangle) = S(\mathbf{int})$

$$\begin{array}{c}
 \text{OFC_INT} \frac{}{\vdash \mathbf{int} : \mathbf{T}} \quad \text{OFC_INT} \frac{}{\alpha_x : S(\mathbf{int}) \vdash \mathbf{int} : \mathbf{T}} \quad \frac{}{\vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (from IH)}} \\
 \text{OFK_SING} \frac{}{\vdash S(\mathbf{int}) : \text{kind}} \quad \text{OFS_DYN} \frac{}{\alpha_x : S(\mathbf{int}) \vdash \langle\langle \mathbf{int} \rangle\rangle : \text{sig}} \quad \frac{}{\alpha_x : S(\mathbf{int}), \alpha : 1 \vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (Lem. 7)}} \\
 \text{OFS_STAT} \frac{}{\vdash \langle\langle S(\mathbf{int}) \rangle\rangle : \text{sig}} \quad \text{OFS_SIGMA} \frac{}{\alpha_x : S(\mathbf{int}) \vdash \Sigma \alpha : \langle\langle \mathbf{int} \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig}} \\
 \text{OFS_SIGMA} \frac{}{\vdash \Sigma \alpha_x : \langle\langle S(\mathbf{int}) \rangle\rangle . \Sigma \alpha : \langle\langle \mathbf{int} \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig}}
 \end{array}$$

Case 3: $L = x :: L', \mathbf{F}_{\mathcal{P}}(x) = f$, where $\vdash f : \mathbf{int} \rightarrow \tau$. We have that $\langle\langle L \rangle\rangle_C = \Sigma \alpha_f : \langle\langle S(\mathbf{int}) \rangle\rangle . \Sigma \alpha_1 : \langle\langle \mathbf{int} \rangle\rangle . \Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C$.

We now look at $\Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C$. Notice that since $\vdash f : \mathbf{int} \rightarrow \tau$, we have that $\vdash \mathbf{int} \rightarrow \tau : \mathbf{T}$. From the ofs_dyn rule, $\vdash \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle : \text{sig}$. Thus, we have that:

$$\begin{array}{c}
 \frac{}{\vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (from IH)}} \\
 \text{OFS_SIGMA} \frac{}{\vdash \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle : \text{sig}} \quad \frac{}{\alpha_2 : 1 \vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (Lem. 7)}} \\
 \frac{}{\vdash \Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig}} \\
 \alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (Lem. 7)}
 \end{array}$$

In addition, by using a reasoning similar to the one in Case 2, we have:

- $\vdash \langle\langle S(\mathbf{int}) \rangle\rangle : \text{sig}$,
- $\alpha_f : S(\mathbf{int}) \vdash \langle\langle \mathbf{int} \rangle\rangle : \text{sig}$.

Therefore, we have that:

$$\begin{array}{c}
 \alpha_f : S(\mathbf{int}) \vdash \langle\langle \mathbf{int} \rangle\rangle : \text{sig} \\
 \text{OFS_SIGMA} \frac{}{\vdash \langle\langle S(\mathbf{int}) \rangle\rangle : \text{sig}} \quad \text{OFS_SIGMA} \frac{}{\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig}} \\
 \frac{}{\vdash \langle\langle S(\mathbf{int}) \rangle\rangle : \text{sig}} \quad \frac{}{\alpha_f : S(\mathbf{int}) \vdash \Sigma \alpha_1 : \langle\langle \mathbf{int} \rangle\rangle . \Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig}} \\
 \text{OFS_SIGMA} \frac{}{\Sigma \alpha_f : \langle\langle S(\mathbf{int}) \rangle\rangle . \Sigma \alpha_1 : \langle\langle \mathbf{int} \rangle\rangle . \Sigma \alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle\rangle . \langle\langle L' \rangle\rangle_C : \text{sig}}
 \end{array}$$

Lemma 14. *For any $L \subseteq \mathbf{V}_{\mathcal{P}}$, it follows that $\vdash \langle\langle L \rangle\rangle_P : \text{sig}$.*

Proof. We prove the lemma by induction on L . We have four cases.

Case 1: $L = []$. We have that $\langle\langle L \rangle\rangle_P = 1$. From the `ofs_one` rule, we have that $\vdash 1 : \text{sig}$.

Case 2: $L = x :: L', x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$. We have that $\langle\langle L \rangle\rangle_P = \Sigma \alpha_x : \langle\langle \mathbf{T} \rangle\rangle. \Sigma \alpha : \langle\langle \alpha_x \rangle\rangle. \langle\langle L' \rangle\rangle_P$. We have the following derivation. Notice that $\text{Fst}(\langle\langle c \rangle\rangle) = 1$ for any c , and $\text{Fst}(\langle\langle \mathbf{T} \rangle\rangle) = \mathbf{T}$.

$$\begin{array}{c} \text{OFS_STAT} \frac{\vdash \mathbf{T} : \text{kind}}{\vdash \langle\langle \mathbf{T} \rangle\rangle : \text{sig}} \quad \text{OFS_DYN} \frac{\text{OFC_VAR} \frac{}{\alpha_x : \mathbf{T} \vdash \alpha_x : \mathbf{T}}}{\alpha_x : \mathbf{T} \vdash \langle\langle \alpha_x \rangle\rangle : \text{sig}} \quad \frac{\vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (from IH)}}{\alpha_x : \mathbf{T}, \alpha : 1 \vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (Lem. 7)}} \\ \text{OFS_SIGMA} \frac{}{\vdash \langle\langle \mathbf{T} \rangle\rangle : \text{sig}} \quad \text{OFS_SIGMA} \frac{\alpha_x : \mathbf{T} \vdash \Sigma \alpha : \langle\langle \alpha_x \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}}{\vdash \Sigma \alpha_x : \langle\langle \mathbf{T} \rangle\rangle. \Sigma \alpha : \langle\langle \alpha_x \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}} \end{array}$$

Case 3: $L = x :: L', \mathbf{F}_{\mathcal{P}}(x) = f$, where $\vdash f : \mathbf{int} \rightarrow \tau$ for some τ . We have that $\langle\langle L \rangle\rangle_P = \Sigma \alpha_f : \langle\langle \mathbf{T} \rangle\rangle. \Sigma \alpha_1 : \langle\langle \alpha_f \rangle\rangle. \Sigma \alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle\rangle. \langle\langle L' \rangle\rangle_P$.

We now look at $\Sigma \alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle\rangle. \langle\langle L' \rangle\rangle_C$. We have that $\alpha_f : \mathbf{T} \vdash \alpha_f : \mathbf{T}$ and $\vdash \tau : \mathbf{T}$ (notice that $\vdash f : \mathbf{int} \rightarrow \tau$ and hence, $\vdash \mathbf{int} \rightarrow \tau : \mathbf{T}$ and hence, $\vdash \tau : \mathbf{T}$). From `ofc_arrow`, we have that $\alpha_f : \mathbf{T} \vdash \alpha_f \rightarrow \tau : \mathbf{T}$. From Lemma 7, it follows that $\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \alpha_f \rightarrow \tau : \mathbf{T}$. From `ofs_dyn` $\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \langle\langle \alpha_f \rightarrow \tau \rangle\rangle : \text{sig}$.

Thus, we have that:

$$\begin{array}{c} \alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \langle\langle \alpha_f \rightarrow \tau \rangle\rangle : \text{sig} \\ \text{OFS_SIGMA} \frac{\frac{\vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (from IH)}}{\alpha_f : \mathbf{T}, \alpha_1 : 1, \alpha_2 : 1 \vdash \langle\langle L' \rangle\rangle_C : \text{sig} \text{ (Lem. 7)}}}{\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \Sigma \alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}} \end{array}$$

In addition, by using a reasoning similar to the one in Case 2, we have that:

- $\vdash \langle\langle \mathbf{T} \rangle\rangle : \text{sig}$,
- $\alpha_f : \mathbf{T} \vdash \langle\langle \alpha_f \rangle\rangle : \text{sig}$.

Therefore, we have that:

$$\text{OFS_SIGMA} \frac{\vdash \langle\langle \mathbf{T} \rangle\rangle : \text{sig} \quad \text{OFS_SIGMA} \frac{\alpha_f : \mathbf{T} \vdash \langle\langle \alpha_f \rangle\rangle : \text{sig} \quad \alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \Sigma \alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}}{\alpha_f : \mathbf{T} \vdash \Sigma \alpha_1 : \langle\langle \alpha_f \rangle\rangle. \Sigma \alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}}}{\Sigma \alpha_f : \langle\langle \mathbf{T} \rangle\rangle. \Sigma \alpha_1 : \langle\langle \alpha_f \rangle\rangle. \Sigma \alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}}$$

Lemma 15. *For any $L \subseteq \mathbf{V}_{\mathcal{P}}$, it follows that $\vdash \langle\langle L \rangle\rangle_C \leq \langle\langle L \rangle\rangle_P : \text{sig}$.*

Proof. We prove the lemma by proving that for any L , $\vdash \langle\langle L \rangle\rangle_C \leq \langle\langle L \rangle\rangle_P : \text{sig}$. We prove this by induction on L . We have four cases.

Case 1: $L = []$. We have that $\langle\langle L \rangle\rangle_C = \langle\langle L \rangle\rangle_P = 1$. From `eqs_refl`, we have that $\vdash 1 \equiv 1 : \text{sig}$ and hence, from the `subs_refl` rule, it follows that $\vdash 1 \leq 1 : \text{sig}$.

Case 2: $L = x :: L'$ and $x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$. We need to prove that $\vdash \Sigma \alpha_x : \langle\langle S(\mathbf{int}) \rangle\rangle. \Sigma \alpha : \langle\langle \mathbf{int} \rangle\rangle. \langle\langle L' \rangle\rangle_C \leq \Sigma \alpha_x : \langle\langle \mathbf{T} \rangle\rangle. \Sigma \alpha : \langle\langle \alpha_x \rangle\rangle. \langle\langle L' \rangle\rangle_C : \text{sig}$.

We first prove that $\alpha_f : \langle S(\mathbf{int}) \rangle \vdash \Sigma \alpha : \langle \mathbf{int} \rangle. \langle L' \rangle_C \leq \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_C : \mathbf{sig}$. To this aim, we prove that $\alpha_x : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \leq \langle \alpha_x \rangle : \mathbf{sig}$.

$$\begin{array}{c} \text{OFC_VAR} \frac{(\alpha_x : S(\mathbf{int}))(\alpha_x) = S(\mathbf{int})}{\alpha_x : S(\mathbf{int}) \vdash \alpha_x : S(\mathbf{int})} \\ \text{EQC_SINGELIM} \frac{\alpha_x : S(\mathbf{int}) \vdash \alpha_x : S(\mathbf{int})}{\alpha_x : S(\mathbf{int}) \vdash \alpha_x \equiv \mathbf{int} : \mathbf{T}} \\ \text{EQC_SYMM} \frac{\alpha_x : S(\mathbf{int}) \vdash \alpha_x \equiv \mathbf{int} : \mathbf{T}}{\alpha_x : S(\mathbf{int}) \vdash \mathbf{int} \equiv \alpha_x : \mathbf{T}} \\ \text{EQS_DYN} \frac{\alpha_x : S(\mathbf{int}) \vdash \mathbf{int} \equiv \alpha_x : \mathbf{T}}{\alpha_x : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \equiv \langle \alpha_x \rangle} \\ \text{SUBS_REFL} \frac{\alpha_x : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \equiv \langle \alpha_x \rangle}{\alpha_x : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \leq \langle \alpha_x \rangle : \mathbf{sig}} \end{array}$$

From IH, we have that $\vdash \langle L' \rangle_C \leq \langle L' \rangle_P : \mathbf{sig}$. From Lemma 7, it follows that $\alpha_x : S(\mathbf{int}), \alpha : 1 \vdash \langle L' \rangle_C \leq \langle L' \rangle_P : \mathbf{sig}$. In addition, from Lemma 14, we have that $\vdash \langle L' \rangle_P : \mathbf{sig}$. From Lemma 7, it follows that $\alpha_x : S(\mathbf{int}), \alpha : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$. Since $\mathbf{fst}(\langle \mathbf{int} \rangle) = \mathbf{fst}(\langle \alpha_x \rangle) = 1$, we have that:

$$\begin{array}{c} \alpha_x : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \leq \langle \alpha_x \rangle : \mathbf{sig} \\ \alpha_x : S(\mathbf{int}), \alpha : 1 \vdash \langle L' \rangle_C \leq \langle L' \rangle_P : \mathbf{sig} \\ \alpha_x : S(\mathbf{int}), \alpha : 1 \vdash \langle L' \rangle_P : \mathbf{sig} \\ \text{SUBS_SIGMA} \frac{\alpha_x : S(\mathbf{int}), \alpha : 1 \vdash \langle L' \rangle_P : \mathbf{sig}}{\alpha_x : S(\mathbf{int}) \vdash \Sigma \alpha : \langle \mathbf{int} \rangle. \langle L' \rangle_C \leq \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_C : \mathbf{sig}} \end{array}$$

We next prove that $\vdash \langle S(\mathbf{int}) \rangle \leq \langle \mathbf{T} \rangle : \mathbf{sig}$. From `ofc_int`, it follows that $\vdash \mathbf{int} : \mathbf{T}$. From `subk_sing_t`, it follows that $\vdash S(\mathbf{int}) \leq \mathbf{T} : \mathbf{kind}$. From `subs_stat`, it follows that $\vdash \langle S(\mathbf{int}) \rangle \leq \langle \mathbf{T} \rangle : \mathbf{sig}$.

We now prove that $\alpha_x : \mathbf{T} \vdash \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P : \mathbf{sig}$. Indeed, we have that $\alpha_x : \mathbf{T} \vdash \langle \alpha_x \rangle : \mathbf{sig}$. From Lemma 14, $\vdash \langle L' \rangle_P : \mathbf{sig}$. From Lemma 7, $\alpha_x : \mathbf{T}, \alpha : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$. From `ofs_sigma`, it follows that $\alpha_x : \mathbf{T} \vdash \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P : \mathbf{sig}$.

Thus, we have that:

$$\begin{array}{c} \vdash \langle S(\mathbf{int}) \rangle \leq \langle \mathbf{T} \rangle : \mathbf{sig} \\ \alpha_x : S(\mathbf{int}) \vdash \Sigma \alpha : \langle \mathbf{int} \rangle. \langle L' \rangle_C \leq \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_C : \mathbf{sig} \\ \alpha_x : \mathbf{T} \vdash \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P : \mathbf{sig} \\ \text{SUBS_SIGMA} \frac{\alpha_x : \mathbf{T} \vdash \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P : \mathbf{sig}}{\vdash \Sigma \alpha_x : \langle S(\mathbf{int}) \rangle. \Sigma \alpha : \langle \mathbf{int} \rangle. \langle L' \rangle_C \leq \Sigma \alpha_x : \langle \mathbf{T} \rangle. \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P} \end{array}$$

Case 3: $L = x :: L'$ and $\mathbf{F}_{\mathcal{P}}(x) = f$, where $\vdash f : \mathbf{int} \rightarrow \tau$ for some τ . We need to prove that

$$\begin{array}{c} \vdash \Sigma \alpha_f : \langle S(\mathbf{int}) \rangle. \Sigma \alpha_1 : \langle \mathbf{int} \rangle. \Sigma \alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \\ \Sigma \alpha_f : \langle \mathbf{T} \rangle. \Sigma \alpha_1 : \langle \alpha_f \rangle. \Sigma \alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}. \end{array}$$

We first prove that $\alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rightarrow \tau \rangle \leq \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$.

$$\begin{array}{c} \text{OFC_VAR} \frac{(\alpha_f : S(\mathbf{int}))(\alpha_f) = S(\mathbf{int})}{\alpha_f : S(\mathbf{int}) \vdash \alpha_f : S(\mathbf{int})} \\ \text{EQC_SINGELIM} \frac{\alpha_f : S(\mathbf{int}) \vdash \alpha_f : S(\mathbf{int})}{\alpha_f : S(\mathbf{int}) \vdash \alpha_f \equiv \mathbf{int} : \mathbf{T}} \\ \text{EQC_SYMM} \frac{\alpha_f : S(\mathbf{int}) \vdash \alpha_f \equiv \mathbf{int} : \mathbf{T}}{\alpha_f : S(\mathbf{int}) \vdash \mathbf{int} \equiv \alpha_f : \mathbf{T}} \\ \text{EQC_ARROW} \frac{\alpha_f : S(\mathbf{int}) \vdash \mathbf{int} \equiv \alpha_f : \mathbf{T}}{\alpha_f : S(\mathbf{int}) \vdash \mathbf{int} \rightarrow \tau \equiv \alpha_f \rightarrow \tau : \mathbf{T}} \\ \text{EQS_DYN} \frac{\alpha_f : S(\mathbf{int}) \vdash \mathbf{int} \rightarrow \tau \equiv \alpha_f \rightarrow \tau : \mathbf{T}}{\alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rightarrow \tau \rangle \equiv \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}} \\ \text{SUBS_REFL} \frac{\alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rightarrow \tau \rangle \equiv \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}}{\alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rightarrow \tau \rangle \leq \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}} \end{array}$$

We next prove that $\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$. We have that

- $\alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rightarrow \tau \rangle \leq \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$ and hence, from Lemma 7, it follows that $\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \langle \mathbf{int} \rightarrow \tau \rangle \leq \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$,
- $\vdash \langle L' \rangle_C \leq \langle L' \rangle_P : \mathbf{sig}$ (from IH) and hence, from Lemma 7, it follows that $\alpha_f : S(\mathbf{int}), \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_C \leq \langle L' \rangle_P : \mathbf{sig}$,
- $\vdash \langle L' \rangle_P : \mathbf{sig}$ (from Lemma 14) and hence, from Lemma 7, it follows that $\alpha_f : S(\mathbf{int}), \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$

Since $\text{Fst}(\langle c \rangle) = 1$ for any c , we have that:

$$\text{SUBS_SIGMA} \frac{\begin{array}{c} \alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \langle \mathbf{int} \rightarrow \tau \rangle \leq \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig} \\ \alpha_f : S(\mathbf{int}), \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_C \leq \langle L' \rangle_P : \mathbf{sig} \\ \alpha_f : S(\mathbf{int}), \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_P : \mathbf{sig} \end{array}}{\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}}$$

We have that:

- $\alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \leq \langle \alpha_f \rangle : \mathbf{sig}$ (as in Case 2),
- $\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$ (as proven above),
- $\alpha_f : S(\mathbf{int}) \vdash \alpha_f \rightarrow \tau : \mathbf{T}$. From the ofs_dyn rule, $\alpha_f : S(\mathbf{int}) \vdash \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$. From Lemma 7, $\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$. From Lemma 14, $\vdash \langle L' \rangle_P : \mathbf{sig}$ and hence, from Lemma 7, $\alpha_f : S(\mathbf{int}), \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$. Since $\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$ and $\alpha_f : S(\mathbf{int}), \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$, from the ofs_sigma rule, it follows that $\alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma\alpha_2. \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$.

Since $\text{Fst}(\langle c \rangle) = 1$ for any c , we have that:

$$\text{SUBDPR} \frac{\begin{array}{c} \alpha_f : S(\mathbf{int}) \vdash \langle \mathbf{int} \rangle \leq \langle \alpha_f \rangle : \mathbf{sig} \\ \alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig} \\ \alpha_f : S(\mathbf{int}), \alpha_1 : 1 \vdash \Sigma\alpha_2. \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig} \end{array}}{\alpha_f : S(\mathbf{int}) \vdash \Sigma\alpha_1 : \langle \mathbf{int} \rangle. \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}}$$

We have that

- $\vdash \langle S(\mathbf{int}) \rangle \leq \langle \mathbf{T} \rangle : \mathbf{sig}$ (as in Case 2), and
- $\alpha_f : S(\mathbf{int}) \vdash \Sigma\alpha_1 : \langle \mathbf{int} \rangle. \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$ (as proven above),
- $\alpha_f : \mathbf{T} \vdash \alpha_f \rightarrow \tau : \mathbf{T}$. From the ofs_dyn rule, $\alpha_f : \mathbf{T} \vdash \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$. From Lemma 7, $\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$. From Lemma 14, $\vdash \langle L' \rangle_P : \mathbf{sig}$ and hence, from Lemma 7, $\alpha_f : \mathbf{T}, \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$. Since $\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \langle \alpha_f \rightarrow \tau \rangle : \mathbf{sig}$ and $\alpha_f : \mathbf{T}, \alpha_1 : 1, \alpha_2 : 1 \vdash \langle L' \rangle_P : \mathbf{sig}$, from the ofs_sigma rule, $\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$. Since $\alpha_f : \mathbf{T} \vdash \alpha_f : \mathbf{T}$, from the ofs_dyn rule, $\alpha_f : \mathbf{T} \vdash \langle \alpha_f \rangle : \mathbf{sig}$. Since $\alpha_f : \mathbf{T} \vdash \langle \alpha_f \rangle : \mathbf{sig}$ and $\alpha_f : \mathbf{T}, \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$, from the ofs_sigma rule, $\alpha_f : \mathbf{T} \vdash \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}$.

From the subs_sigma rule, we have that

$$\begin{aligned} \vdash \Sigma\alpha_f : \langle S(\mathbf{int}) \rangle. \Sigma\alpha_1 : \langle \mathbf{int} \rangle. \Sigma\alpha_2 : \langle \mathbf{int} \rightarrow \tau \rangle. \langle L' \rangle_C \leq \\ \Sigma\alpha_f : \langle \mathbf{T} \rangle. \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P : \mathbf{sig}. \end{aligned}$$

Lemma 16 (Pitts closure). *For any x , f and a in the policy, it follows that R_x , R_f and $R_{f \circ a}$ are Pitts closed.*

Proof. We consider R_x first. The proof of this case is trivial since any v_1 and v_2 s.t. $\vdash v_i : \mathbf{int}$, we have that $\langle v_1, v_2 \rangle \in R_x$.

We next consider R_f , where $\vdash f : \mathbf{int} \rightarrow \tau$. We consider $x : \mathbf{int} \vdash f x$. We have that x is active in $f x$. We now consider arbitrary v_1 and v_2 s.t. $\langle v_1, v_2 \rangle$ in R_f . From the definition of R_f , we have that $\langle f v_1, f v_2 \rangle \in \llbracket \tau \rrbracket_\emptyset^{\text{ev}}$. Since $\vdash \tau : \mathbf{T}$, from Lemma 9, we have that $\langle \neg, \neg, \llbracket \tau \rrbracket_\emptyset, \llbracket \tau \rrbracket_\emptyset \rangle \in \llbracket \mathbf{T} \rrbracket_\emptyset$ (notice that $\emptyset \in \llbracket \cdot \rrbracket^{\text{full}}$). Therefore, $\llbracket \tau \rrbracket_\emptyset$ is Pitts closed, that is $\llbracket \tau \rrbracket_\emptyset = \llbracket \tau \rrbracket_\emptyset^{\text{st}}$. Since $\langle f v_1, f v_2 \rangle \in \llbracket \tau \rrbracket_\emptyset^{\text{ev}}$, it follows that $\langle f v_1, f v_2 \rangle \in \llbracket \tau \rrbracket_\emptyset^{\text{stev}}$.

We have proven that:

- x is active in $f x$,
- for all $\langle v_1, v_2 \rangle \in R_f$, $\langle (f x)[x \mapsto v_1], (f x)[x \mapsto v_2] \rangle \in \llbracket \tau \rrbracket_\emptyset^{\text{stev}}$.

From Lemma 8, for all $\langle w_1, w_2 \rangle \in R_f^{\text{st}}$, we have that $\langle f w_1, f w_2 \rangle \in \llbracket \tau \rrbracket_\emptyset^{\text{stev}} = \llbracket \tau \rrbracket_\emptyset^{\text{ev}}$. From the definition of R_f , we have that $\langle w_1, w_2 \rangle \in R_f$.

Lemma 17. *For any $L \subseteq \mathbf{V}_{\mathcal{P}}$ and $\rho \in |L|_{\mathcal{P}}$, it follows that*

- $\vdash_P \rho_L(m_{\mathcal{P}}) : \langle L \rangle_C$ and $\vdash_P \rho_R(m_{\mathcal{P}}) : \langle L \rangle_C$, and
- $\vdash_P \rho_L(m_{\mathcal{P}}) : \langle L \rangle_P$ and $\vdash_P \rho_R(m_{\mathcal{P}}) : \langle L \rangle_P$.

Proof. The first part of the Lemma 17 is from the definition of $\rho \in |L|_{\mathcal{P}}$. The second part follows from the first part, Lemma 15, and the subsumption rule.

Lemma 18. *Suppose that $L \subseteq \mathbf{V}_{\mathcal{P}}$, $\rho \in |L|_{\mathcal{P}}$, $\rho(\alpha_{\mathcal{P}}) = \langle c_1, c_2, Q \rangle$, $k = \text{Fst}(\langle L \rangle_P)$. It follows that:*

- $\vdash k : \text{kind}$,
- $\vdash c_1 : \rho_L(k)$, $\vdash c_2 : \rho_R(k)$, and
- $\langle c_1, c_2, Q, Q \rangle \in \llbracket k \rrbracket_\rho$.

Proof. We prove this lemma by induction on L , using the definition of $\rho \in |L|_{\mathcal{P}}$.

Case 1: $L = \square$. We have that $\sigma = 1$ and $k = 1$, $\rho(\alpha_{\mathcal{P}}) = \langle \star, \star, \langle \rangle \rangle$. We can easily check that $\vdash 1 : \text{kind}$, $\vdash \star : 1$, and $\langle \star, \star, \langle \rangle, \langle \rangle \rangle \in \llbracket 1 \rrbracket_\rho$.

Case 2: $L = x :: L'$, where $x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$. We have that

- $\sigma = \Sigma\alpha_x : \langle \mathbf{T} \rangle. \Sigma\alpha : \langle \alpha_x \rangle. \langle L' \rangle_P$, and
- $k = \Sigma\alpha_x : \mathbf{T}. \Sigma\alpha : 1. \text{Fst}(\langle L' \rangle_P)$,
- $Q = \langle R_x, \langle \rangle, Q' \rangle$,
- $\rho(\alpha_{\mathcal{P}}) = \langle \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle, \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle, \langle R_x, \langle \rangle, Q' \rangle \rangle$,
- $\rho' \in |L'|_{\mathcal{P}}$, where $\rho'(\alpha_{\mathcal{P}}) = \langle c'_1, c'_2, Q' \rangle$.

We need to prove that:

- $\vdash \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$,
- $\vdash \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle : \rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$,
- $\vdash \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle : \rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$,
- $\langle \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle, \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle, \langle R_x, \langle \langle \rangle, Q' \rangle \rangle \rangle \in \llbracket \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_\rho$.

We first prove that $\vdash \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. From IH, we have that $\vdash \text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. From Lemma 7, $\alpha : 1 \vdash \text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. From rule `ofk_sigma`, $\vdash \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. From Lemma 7, $\alpha_x : \mathbb{T} \vdash \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. From rule `ofk_sigma`, $\vdash \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$.

We next prove that $\vdash \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle : \rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$ and $\vdash \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle : \rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$. Since their proofs are similar, we only prove here $\vdash \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle : \rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$. Notice that since $\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P)$ is a closed kind (as proven above), we have that $\rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P)) = \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P)$.

We have the following derivations. Notice that

- $\vdash \mathbf{int} : \mathbb{T}$,
- if k' is a closed kind, then $k'[\beta \mapsto c'] = k'$ for any β and c' ,
- $\vdash c'_1 : \rho'_L(\text{Fst}(\langle\langle L' \rangle\rangle_P))$ (from IH) and hence, $\vdash c'_1 : \text{Fst}(\langle\langle L' \rangle\rangle_P)$ (since from IH, $\text{Fst}(\langle\langle L' \rangle\rangle_P)$ is a closed kind). Thus, $\vdash c'_1 : \text{Fst}(\langle\langle L' \rangle\rangle_P)[\alpha \mapsto \star]$.
- $\vdash \text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$ (from IH) and hence, from Lemma 7, $\alpha : 1 \vdash \text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$,
- $\vdash \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$ (from the `ofk_sigma` rule and $\alpha : 1 \vdash \text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$) and hence $\alpha_x : \mathbb{T} \vdash \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$ (from Lemma 7).

$$\text{OFC_PAIR} \frac{\vdash \star : 1 \quad \vdash c'_1 : \text{Fst}(\langle\langle L' \rangle\rangle_P)[\alpha \mapsto \star] \quad \alpha : 1 \vdash \text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}}{\vdash \langle \star, c'_1 \rangle : \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P)}$$

$$\text{OFC_PAIR} \frac{\vdash \mathbf{int} : \mathbb{T} \quad \vdash \langle \star, c'_1 \rangle : \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P)[\alpha_x \mapsto \mathbf{int}] \quad \alpha_x : \mathbb{T} \vdash \Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}}{\vdash \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle : \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P)}$$

We now prove that $\langle \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle, \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle, \langle R_x, \langle \langle \rangle, Q' \rangle \rangle \rangle \in \llbracket \Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_\rho$. As proven above, we have that $\vdash \langle \mathbf{int}, \langle \star, c'_1 \rangle \rangle : \rho_L(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$ and $\vdash \langle \mathbf{int}, \langle \star, c'_2 \rangle \rangle : \rho_R(\Sigma\alpha_x : \mathbb{T}.\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$. We now need to prove that:

- $\langle R_x, \langle \langle \rangle, Q' \rangle \rangle \in \text{PreCand}_{\text{simp}(\langle\langle L \rangle\rangle_P)}$,
- $\langle \mathbf{int}, \mathbf{int}, R_x, R_x \rangle \in \llbracket \mathbb{T} \rrbracket_\rho$,
- $\vdash \langle \star, c'_1 \rangle : \rho_{1L}(\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$,
- $\vdash \langle \star, c'_2 \rangle : \rho_{1R}(\Sigma\alpha : 1.\text{Fst}(\langle\langle L' \rangle\rangle_P))$,
- $\langle \star, \star, \langle \rangle, \langle \rangle \rangle \in \llbracket 1 \rrbracket_{\rho_1}$,
- $\vdash c'_1 : \rho_{2L}(\text{Fst}(\langle\langle L' \rangle\rangle_P))$,
- $\vdash c'_2 : \rho_{2R}(\text{Fst}(\langle\langle L' \rangle\rangle_P))$,

$$- \langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho_2},$$

where $\rho_1 = \rho, \alpha_x \mapsto \langle \mathbf{int}, \mathbf{int}, R_x \rangle$ and $\rho_2 = \rho_1, \alpha \mapsto \langle \star, \star, \langle \rangle \rangle$.

We have that:

- We have that $R_x \in \mathbf{PreCand}_T$, $\langle \rangle \in \mathbf{PreCand}_1$. From IH, $\langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \langle\langle L' \rangle\rangle_P \rrbracket_{\rho'}$ and hence, $Q' \in \mathbf{PreCand}_{\mathbf{simp}(\mathbf{Fst}(\langle\langle L' \rangle\rangle_P))}$ (notice that $\mathbf{simp}(\mathbf{Fst}(\langle\langle L' \rangle\rangle_P)) = \mathbf{Fst}(\langle\langle L' \rangle\rangle_P)$ since there is no singleton kind in $\mathbf{Fst}(\langle\langle L' \rangle\rangle_P)$).
- From Lemma 16, $R_x = R_x^{\text{st}}$. Thus, $\langle \mathbf{int}, \mathbf{int}, R_x, R_x \rangle \in \llbracket \mathbf{T} \rrbracket_{\rho}$.
- As shown in the first derivation in the proof above, we have that $\vdash \langle \star, c'_1 \rangle : \Sigma\alpha : 1.\mathbf{Fst}(\langle\langle L' \rangle\rangle_P)$ and hence, $\vdash \langle \star, c'_1 \rangle : \rho_{1L}(\Sigma\alpha : 1.\mathbf{Fst}(\langle\langle L' \rangle\rangle_P))$ (since $\Sigma\alpha : 1.\mathbf{Fst}(\langle\langle L' \rangle\rangle_P)$ is closed)
- Similarly, it follows that $\vdash \langle \star, c'_2 \rangle : \rho_{1R}(\Sigma\alpha : 1.\mathbf{Fst}(\langle\langle L' \rangle\rangle_P))$.
- Since $L' \subseteq \mathbf{V}_{\mathcal{P}}$ and $\rho' \in |L'|_{\mathcal{P}}$, from IH, $\vdash c'_1 : \rho'_L(k')$, where $\rho' \in |L'|_{\mathcal{P}}$ and $k' = \mathbf{Fst}(\langle\langle L' \rangle\rangle_P)$. Also from IH, $\vdash k'$. Thus, $\vdash c'_1 : k'$ and hence, $\vdash c'_1 : \rho_{2L}(k')$.
- Similarly, $\vdash c'_1 : \rho_{2R}(k')$.
- Since $L' \subseteq \mathbf{V}_{\mathcal{P}}$ and $\rho' \in |L'|_{\mathcal{P}}$, from IH, $\langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho'}$, where $\rho' \in |L'|_{\mathcal{P}}$. Also from IH, $\vdash \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. Since $\langle \rho', \rho_2 \rangle \in \llbracket \cdot \rrbracket$, from Lemma 9, we have that $\llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho'} = \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho_2}$. Therefore, $\langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho_2}$.

Case 3: $L = x :: L'$ and $\mathbf{F}_{\mathcal{P}}(x) = f$, where $\vdash f : \mathbf{int} \rightarrow \tau$. We have that $\langle\langle L \rangle\rangle_P = \Sigma\alpha_f : \langle \mathbf{T} \rangle . \alpha_1 : \langle \alpha_f \rangle . \alpha_2 : \langle \alpha_f \rightarrow \tau_f \rangle . \langle\langle L' \rangle\rangle_P$, $k = \Sigma\alpha_f : \mathbf{T} . \alpha_1 : 1 . \alpha_2 : 1.\mathbf{Fst}(\langle\langle L' \rangle\rangle_P)$, and $\rho'(\alpha_{\mathcal{P}}) = \langle c'_1, c'_2, Q' \rangle$, and

$$\rho(\alpha_{\mathcal{P}}) = \langle \langle \mathbf{int}, \langle \star, \langle \star, c'_1 \rangle \rangle \rangle, \langle \mathbf{int}, \langle \star, \langle \star, c'_2 \rangle \rangle \rangle, \langle R_f, \langle \langle \rangle, \langle \langle \rangle, Q' \rangle \rangle \rangle \rangle,$$

As in Case 2, we have that $\vdash k : \text{kind}$ and $\vdash \langle \mathbf{int}, \langle \star, \langle \star, c'_1 \rangle \rangle \rangle : k$. We now prove that $\langle c_1, c_2, Q, Q \rangle \in \llbracket \Sigma\alpha_f : \mathbf{T} . \alpha_1 : 1 . \alpha_2 : 1.\mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho}$. That is we need to prove that:

- $\langle R_f, \langle \langle \rangle, \langle \langle \rangle, Q' \rangle \rangle \rangle \in \mathbf{PreCand}_{\mathbf{simp}(\langle\langle L \rangle\rangle_P)}$,
- $\langle \mathbf{int}, \mathbf{int}, R_f, R_f \rangle \in \llbracket \mathbf{T} \rrbracket_{\rho}$,
- $\langle \star, \star, \langle \rangle, \langle \rangle \rangle \in \llbracket \mathbf{1} \rrbracket_{\rho_1}$, where $\rho_1 = \rho, \alpha_f \mapsto \langle \mathbf{int}, \mathbf{int}, R_f \rangle$,
- $\langle \star, \star, \langle \rangle, \langle \rangle \rangle \in \llbracket \mathbf{1} \rrbracket_{\rho_2}$, where $\rho_2 = \rho_1, \alpha_1 \mapsto \langle \star, \star, \langle \rangle \rangle$,
- $\langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho_3}$, where $\rho_3 = \rho_2, \alpha_2 \mapsto \langle \star, \star, \langle \rangle \rangle$.

The first item can be easily verified (as in Case 2). From Lemma 16, R_f is Pitts closed and hence, $\langle \mathbf{int}, \mathbf{int}, R_f, R_f \rangle \in \llbracket \mathbf{T} \rrbracket_{\rho}$. We can easily verify that $\langle \star, \star, \langle \rangle, \langle \rangle \rangle \in \llbracket \mathbf{1} \rrbracket_{\rho_1}$ and $\langle \star, \star, \langle \rangle, \langle \rangle \rangle \in \llbracket \mathbf{1} \rrbracket_{\rho_2}$.

We have that $L' \subseteq \mathbf{V}_{\mathcal{P}}$ and $\rho' \in |L'|_{\mathcal{P}}$ (since $\rho \in |L|_{\mathcal{P}}$), from IH, we have that $\langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho'}$. Also from IH, we have that $\vdash \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) : \text{kind}$. Since $\langle \rho', \rho_3 \rangle \in \llbracket \cdot \rrbracket$, from Lemma 9, we have that $\llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho'} = \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho_3}$. Therefore, we have that $\langle c'_1, c'_2, Q', Q' \rangle \in \llbracket \mathbf{Fst}(\langle\langle L' \rangle\rangle_P) \rrbracket_{\rho_3}$.

Lemma 10 (in §G) If $\rho_1 \models^{\text{full}} \mathcal{P}$, $\rho_2 \models^{\text{full}} \mathcal{P}$, and $\Gamma_{\mathcal{P}}^{\mathcal{P}} \vdash \tau : \mathbf{T}$, then $\llbracket \tau \rrbracket_{\rho_1} = \llbracket \tau \rrbracket_{\rho_2}$.

Proof. From the definition of $\rho_i \in |\mathbf{V}_{\mathcal{P}}|_{\mathcal{P}}$, we have that $\rho_1(\alpha_{\mathcal{P}}) = \rho_2(\alpha_{\mathcal{P}}) = \langle c_1, c_2, Q \rangle$ for some c_1, c_2 , and Q . From Lemma 18 and the definition of constructor equivalence, we have that:

- $\vdash c_1 \equiv c_1 : \rho_{1L}(\mathbf{Fst}(\sigma_{\mathcal{P}})), \vdash c_2 \equiv c_2 : \rho_{1R}(\mathbf{Fst}(\sigma_{\mathcal{P}}))$, and
- $\langle c_1, c_2, Q, Q \rangle \in \llbracket \mathbf{Fst}(\sigma_{\mathcal{P}}) \rrbracket_{\rho_1}$.

In other words, $\langle \rho_1, \rho_2 \rangle \in \llbracket \alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma_{\mathcal{P}} \rrbracket$.

Since $I_{\mathcal{P}}^{\mathcal{P}} \vdash \tau : \mathbf{T}$, from Lemma 9, we have that $\langle _, _, \llbracket \tau \rrbracket_{\rho_1}, \llbracket \tau \rrbracket_{\rho_2} \rangle \in \llbracket \mathbf{T} \rrbracket_{\rho_1}$. From the definition of $\llbracket \mathbf{T} \rrbracket_{\rho}$ (see Fig. 23), it follows that $\llbracket \tau \rrbracket_{\rho_1} = \llbracket \tau \rrbracket_{\rho_2}$.

Lemma 11 (in §G). Suppose that $\rho \models^{\text{full}} \mathcal{P}$. It follows that $\rho \in \llbracket I_{\mathcal{P}}^{\mathcal{P}} \rrbracket^{\text{full}}$.

Proof. We need to prove that $\rho \models^{\text{full}} \llbracket \alpha_{\mathcal{P}}/m_{\mathcal{P}} : \langle \mathbf{V}_{\mathcal{P}} \rangle_P \rrbracket$. We claim that for any $L \subseteq \mathbf{V}_{\mathcal{P}}$ and any $\rho \in |L|_{\mathcal{P}}$, it follows that $\rho \models^{\text{full}} \llbracket \alpha_{\mathcal{P}}/m_{\mathcal{P}} : \langle L \rangle_P \rrbracket$. Then the proof follows directly from the claim.

We now prove the claim. Suppose that $\rho(\alpha_{\mathcal{P}}) = \langle c_1, c_2, Q \rangle$ and $\rho(m_{\mathcal{P}}) = \langle V_1, V_2 \rangle$. From Lemma 18, we have that:

- $\vdash c_1 : \rho_L(k), \vdash c_2 : \rho_R(k)$ where $k = \mathbf{Fst}(\langle L \rangle_P)$, and hence, it follows that
- $\vdash c_1 \equiv c_1 : \rho_L(k)$ and $\vdash c_2 \equiv c_2 : \rho_R(k)$
- $\langle c_1, c_2, Q, Q \rangle \in \llbracket k \rrbracket_{\rho}$.

Therefore, $\langle \rho, \rho \rangle \in \llbracket \alpha_{\mathcal{P}}/m_{\mathcal{P}} : \langle L \rangle_P \rrbracket$. Thus, we only need to prove two following items:

- $\rho(\alpha_{\mathcal{P}}) = \langle \mathbf{Fst}(V_1), \mathbf{Fst}(V_2), Q \rangle$,
- $\langle V_1, V_2, Q \rangle \in \llbracket \langle L \rangle_P \rrbracket_{\rho}$.

We prove these two items by induction on L , using the definition of $\rho \in |L|_{\mathcal{P}}$. We have four cases.

Case 1: $L = \square$. We have that $\sigma = \langle L \rangle_P = 1$, $\rho(m_{\mathcal{P}}) = \langle \star, \star \rangle$ and $\rho(\alpha_{\mathcal{P}}) = \langle \star, \star, \langle \rangle \rangle$. In other words, $V_1 = V_2 = \star$ and $Q = \langle \rangle$. Since $\vdash \mathbf{Fst}(\star) \gg \star$, from the definition of $\llbracket 1 \rrbracket_{\rho}$, we have that $\langle V_1, V_2, Q \rangle \in \llbracket \sigma \rrbracket_{\rho}$.

Case 2: $L = x :: L', x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$. We have that:

$$\begin{aligned} \rho(m_{\mathcal{P}}) &= \langle \langle \mathbf{int} \rangle, \langle \langle v_1 \rangle, V_1' \rangle, \langle \langle \mathbf{int} \rangle, \langle \langle v_2 \rangle, V_2' \rangle \rangle \rangle, \\ \rho(\alpha_{\mathcal{P}}) &= \langle \langle \mathbf{int}, \langle \star, c_1' \rangle \rangle, \langle \mathbf{int}, \langle \star, c_2' \rangle \rangle, \langle R_x, \langle \langle \rangle, Q' \rangle \rangle \rangle, \end{aligned}$$

and $\langle v_1, v_2 \rangle \in R_x$, and $\rho' \in |L'|_{\mathcal{P}}$, where $\rho'(m_{\mathcal{P}}) = \langle V_1', V_2' \rangle$, $\rho'(\alpha_{\mathcal{P}}) = \langle c_1', c_2', Q' \rangle$. We also have that $\rho = \Sigma \alpha_x : \langle \mathbf{T} \rangle. \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P$.

Since $\rho'(m_{\mathcal{P}}) = \langle V_1', V_2' \rangle$, $\rho'(\alpha_{\mathcal{P}}) = \langle c_1', c_2', Q' \rangle$, and $\rho' \in |L'|_{\mathcal{P}}$, from IH, we have that:

- $c_1' = \mathbf{Fst}(V_1'), c_2' = \mathbf{Fst}(V_2')$, and
- $\langle V_1', V_2', Q' \rangle \in \llbracket \langle L' \rangle_P \rrbracket_{\rho'}$.

Therefore, we have that $\mathbf{Fst}(V_1) = \langle \mathbf{int}, \langle \star, \mathbf{Fst}(V_1') \rangle \rangle$, $\mathbf{Fst}(V_2) = \langle \mathbf{int}, \langle \star, \mathbf{Fst}(V_2') \rangle \rangle$. In other words, $\rho(\alpha_{\mathcal{P}}) = \langle \mathbf{Fst}(V_1), \mathbf{Fst}(V_2), \langle R_x, \langle \langle \rangle, Q' \rangle \rangle \rangle$.

We now need to prove that $\langle \langle \langle \mathbf{int} \rangle, \langle \langle v_1 \rangle, V_1' \rangle \rangle, \langle \langle \mathbf{int} \rangle, \langle \langle v_2 \rangle, V_2' \rangle \rangle, \langle R_x, \langle \langle \rangle, Q' \rangle \rangle \rangle \in \llbracket \Sigma \alpha_x : \langle \mathbf{T} \rangle. \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P \rrbracket_{\rho}$.

From the definition of $\llbracket \Sigma \alpha_x : \langle \mathbf{T} \rangle. \Sigma \alpha : \langle \alpha_x \rangle. \langle L' \rangle_P \rrbracket_{\rho}$, we need to prove that:

- $\vdash_1 V_1 : \rho_L(\langle\langle L \rangle\rangle_P)$,
- $\vdash_1 V_2 : \rho_R(\langle\langle L \rangle\rangle_P)$,
- $\langle\langle \mathbf{int} \rangle\rangle, \langle\langle \mathbf{int} \rangle\rangle, R_x \in \llbracket \langle\langle \mathbf{T} \rangle\rangle \rrbracket_\rho$,
- $\langle\langle \langle v_1 \rangle, V'_1 \rangle, \langle \langle v_2 \rangle, V'_2 \rangle, \langle \langle \rangle, Q' \rangle \rangle \in \llbracket \Sigma\alpha : \langle\langle \alpha_x \rangle \rangle. \langle\langle L' \rangle\rangle_P \rrbracket_{\rho_1}$, where $\rho_1 = \rho, \alpha_x \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_x \rangle\rangle$:
 - $\vdash_1 \langle\langle v_1 \rangle, V'_1 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$ (since $\Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$ is a closed signature),
 - $\vdash_1 \langle\langle v_2 \rangle, V'_2 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$ (since $\Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$ is a closed signature),
 - $\langle\langle v_1 \rangle, \langle v_2 \rangle, \langle \rangle \rangle \in \llbracket \langle\langle \alpha_x \rangle \rangle \rrbracket_{\rho_1}$,
 - $\langle V'_1, V'_2, Q' \rangle \in \llbracket \langle\langle L' \rangle\rangle_P \rrbracket_{\rho_2}$, where $\rho_2 = \rho, \alpha_x \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_x \rangle\rangle, \alpha \mapsto \langle\langle \mathbf{fst}(\langle v'_1 \rangle), \mathbf{fst}(\langle v'_2 \rangle), \langle \rangle) \rangle$.

These items are proven as below.

- From Lemma 17, $\vdash_P V_1 : \langle\langle L \rangle\rangle_P$ and hence $\vdash_1 V_1 : \langle\langle L \rangle\rangle_P$. From Lemma 14, $\vdash \langle\langle L \rangle\rangle_P : \mathbf{sig}$. Thus, $\vdash_1 V_1 : \rho_L(\langle\langle L \rangle\rangle_P)$.
- Similarly, we have that $\vdash_1 V_2 : \rho_R(\langle\langle L \rangle\rangle_P)$.
- From Lemma 16, we have that $\langle\langle \mathbf{int}, \mathbf{int}, R_x, R_x \rangle\rangle \in \llbracket \langle\langle \mathbf{T} \rangle\rangle \rrbracket_\rho$ and hence, $\langle\langle \mathbf{int} \rangle\rangle, \langle\langle \mathbf{int} \rangle\rangle, R_x \in \llbracket \langle\langle \mathbf{T} \rangle\rangle \rrbracket_\rho$.
- We have that $\vdash_P \langle v_1 \rangle : \langle\langle \mathbf{int} \rangle \rangle$. From Lemma 17, $\vdash_P V'_1 : \langle\langle L' \rangle\rangle_C$. Since $\alpha \notin FV(\langle\langle L' \rangle\rangle_C)$, from the ofm_pair rule, $\vdash_P \langle\langle v_1 \rangle, V'_1 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_C$. We have that $\vdash \langle\langle \mathbf{int} \rangle \rangle \leq \langle\langle \mathbf{int} \rangle \rangle : \mathbf{sig}$, $\alpha : 1 \vdash \langle\langle L' \rangle\rangle_C \leq \langle\langle L' \rangle\rangle_P : \mathbf{sig}$, and $\alpha : 1 \vdash \langle\langle L' \rangle\rangle_P : \mathbf{sig}$. Thus,

$$\text{SUBS_SIGMA} \frac{\vdash \langle\langle \mathbf{int} \rangle \rangle \leq \langle\langle \mathbf{int} \rangle \rangle : \mathbf{sig} \quad \alpha : 1 \vdash \langle\langle L' \rangle\rangle_C \leq \langle\langle L' \rangle\rangle_P : \mathbf{sig} \quad \alpha : 1 \vdash \langle\langle L' \rangle\rangle_P : \mathbf{sig}}{\vdash \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_C \leq \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P : \mathbf{sig}}$$

Since $\vdash_P \langle\langle v_1 \rangle, V'_1 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_C$, from ofm_subsume, $\vdash_P \langle\langle v_1 \rangle, V'_1 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$. From the forgetful rule, $\vdash_1 \langle\langle v_1 \rangle, V'_1 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$.

- Similarly, we have that $\vdash_1 \langle\langle v_2 \rangle, V'_2 \rangle : \Sigma\alpha : \langle\langle \mathbf{int} \rangle \rangle. \langle\langle L' \rangle\rangle_P$,
- From the requirement on v_1 and v_2 in $\rho \in |L|_{\mathcal{P}}$, and the definition of $\llbracket \alpha_x \rrbracket_{\rho, \alpha_x \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_x \rangle\rangle}$, we have that $\langle v_1, v_2 \rangle \in \llbracket \alpha_x \rrbracket_{\rho, \alpha_x \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_x \rangle\rangle}$ and hence, $\langle\langle v_1 \rangle, \langle v_2 \rangle, \langle \rangle \rangle \in \llbracket \langle\langle \alpha_x \rangle \rangle \rrbracket_{\rho, \alpha_x \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_x \rangle\rangle} = \llbracket \langle\langle \alpha_x \rangle \rangle \rrbracket_{\rho_1}$.
- From IH, we have that $\langle V'_1, V'_2, Q' \rangle \in \llbracket \langle\langle L' \rangle\rangle_P \rrbracket_{\rho'}$. Since $\vdash \langle\langle L' \rangle\rangle_P : \mathbf{sig}$, and $\langle \rho', \rho_2 \rangle \in \llbracket \cdot \rrbracket$, from Lemma 9, we have that $\llbracket \langle\langle L' \rangle\rangle_P \rrbracket_{\rho'} = \llbracket \langle\langle L' \rangle\rangle_P \rrbracket_{\rho_2}$. Therefore, we have that $\langle V'_1, V'_2, Q' \rangle \in \llbracket \langle\langle L' \rangle\rangle_P \rrbracket_{\rho_2}$.

Case 3: $L = x :: L', \mathbf{F}_{\mathcal{P}}(x) = f$, where $\vdash f : \mathbf{int} \rightarrow \tau$. We have that:

$$\begin{aligned} \rho(m_{\mathcal{P}}) &= \langle\langle \langle\langle \mathbf{int} \rangle\rangle, \langle\langle v_1 \rangle, \langle\langle f \rangle, V'_1 \rangle \rangle \rangle, \langle\langle \langle\langle \mathbf{int} \rangle\rangle, \langle\langle v_2 \rangle, \langle\langle f \rangle, V'_2 \rangle \rangle \rangle \rangle, \\ \rho(\alpha_{\mathcal{P}}) &= \langle\langle \mathbf{int}, \langle \star, \langle \star, c'_1 \rangle \rangle \rangle, \langle\langle \mathbf{int}, \langle \star, \langle \star, c'_2 \rangle \rangle \rangle, \langle R_f, \langle \rangle, \langle \langle \rangle, Q' \rangle \rangle \rangle, \end{aligned}$$

and $\langle v_1, v_2 \rangle \in R_f$, and $\rho' \in |L'|_{\mathcal{P}}$, where $\rho'(m_{\mathcal{P}}) = \langle V'_1, V'_2 \rangle$ and $\rho'(\alpha_{\mathcal{P}}) = \langle c'_1, c'_2, Q' \rangle$. We also have that $\sigma = \Sigma\alpha_f : \langle\langle \mathbf{T} \rangle\rangle. \Sigma\alpha_1 : \langle\langle \alpha_f \rangle \rangle. \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle\rangle_P$

The proof that $\rho(\alpha_{\mathcal{P}}) = \langle\langle \mathbf{fst}(V_1), \mathbf{fst}(V_2), Q \rangle\rangle$ is similar to the one in Case 2. We now prove that $\langle V_1, V_2, Q \rangle \in \llbracket \langle\langle L \rangle\rangle_P \rrbracket_\rho$. We need to prove that:

- $\vdash_1 V_1 : \rho_L(\langle\langle L \rangle\rangle_P)$,
- $\vdash_1 V_2 : \rho_R(\langle\langle L \rangle\rangle_P)$,
- $\langle\langle \mathbf{int} \rangle, \langle\langle \mathbf{int} \rangle, R_f \rangle \in \llbracket \langle\langle \mathbf{T} \rangle \rangle \rrbracket_\rho$
- $\langle\langle \langle v_1 \rangle, \langle\langle f \rangle, V_1' \rangle \rangle, \langle\langle v_2 \rangle, \langle\langle f \rangle, V_2' \rangle \rangle, \langle\langle \rangle, \langle\langle \rangle, Q' \rangle \rangle \in \llbracket \Sigma\alpha_1 : \langle\langle \alpha_f \rangle \rangle. \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_1}$,
where $\rho_1 = \rho, \alpha_f \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_f \rangle \rangle$

We have that:

- From Lemma 17, $\vdash_P V_1 : \langle\langle L \rangle\rangle_P$ and hence $\vdash_1 V_1 : \langle\langle L \rangle\rangle_P$. From Lemma 14,
 $\vdash \langle\langle L \rangle\rangle_P : \mathbf{sig}$. Thus, $\vdash_1 V_1 : \rho_L(\langle\langle L \rangle\rangle_P)$.
- Similarly, we have that $\vdash_1 V_2 : \rho_R(\langle\langle L \rangle\rangle_P)$.
- From Lemma 16, R_f is Pitts closed. Thus, $\langle\langle \mathbf{int} \rangle, \langle\langle \mathbf{int} \rangle, R_f \rangle \in \llbracket \langle\langle \mathbf{T} \rangle \rangle \rrbracket_\rho$.

We now prove that

$$\langle\langle \langle v_1 \rangle, \langle\langle f \rangle, V_1' \rangle \rangle, \langle\langle v_2 \rangle, \langle\langle f \rangle, V_2' \rangle \rangle, \langle\langle \rangle, \langle\langle \rangle, Q' \rangle \rangle \in \llbracket \Sigma\alpha_1 : \langle\langle \alpha_f \rangle \rangle. \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_1}.$$

We need to prove that:

- $\vdash_1 \langle\langle v_1 \rangle, \langle\langle f \rangle, V_1' \rangle \rangle : \rho_{1L}(\Sigma\alpha_1 : \langle\langle \alpha_f \rangle \rangle. \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P)$ and hence, \vdash_1
 $\langle\langle v_1 \rangle, \langle\langle f \rangle, V_1' \rangle \rangle : \Sigma\alpha_1 : \langle\langle \mathbf{int} \rangle \rangle. \Sigma\alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$,
- $\vdash_1 \langle\langle v_2 \rangle, \langle\langle f \rangle, V_2' \rangle \rangle : \rho_{1R}(\Sigma\alpha_1 : \langle\langle \alpha_f \rangle \rangle. \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P)$ and hence, \vdash_1
 $\langle\langle v_2 \rangle, \langle\langle f \rangle, V_2' \rangle \rangle : \Sigma\alpha_1 : \langle\langle \mathbf{int} \rangle \rangle. \Sigma\alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$,
- $\langle\langle v_1 \rangle, \langle\langle v_2 \rangle, \langle\langle \rangle \rangle \rangle \in \llbracket \langle\langle \alpha_f \rangle \rangle \rrbracket_{\rho_1}$,
- $\langle\langle \langle f \rangle, V_1' \rangle, \langle\langle f \rangle, V_2' \rangle, \langle\langle \rangle, Q' \rangle \rangle \in \llbracket \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_2}$, where $\rho_2 =$
 $\rho_1, \alpha_1 \mapsto \langle\langle \star, \star, \langle\langle \rangle \rangle \rangle$.

We have that:

- By using similar reasoning as in Case 2, $\vdash_1 \langle\langle v_1 \rangle, \langle\langle f \rangle, V_1' \rangle \rangle : \Sigma\alpha_1 : \langle\langle \mathbf{int} \rangle \rangle. \Sigma\alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$.
- Similarly, $\vdash_1 \langle\langle v_2 \rangle, \langle\langle f \rangle, V_2' \rangle \rangle : \Sigma\alpha_1 : \langle\langle \mathbf{int} \rangle \rangle. \Sigma\alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$.
- From the definition of $\rho \in |L|_P$, $\langle v_1, v_2 \rangle \in R_f = \llbracket \alpha_f \rrbracket_{\rho, \alpha_f \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_f \rangle \rangle}$. Thus,
 $\langle\langle v_1 \rangle, \langle\langle v_2 \rangle, \langle\langle \rangle \rangle \rangle \in \llbracket \langle\langle \alpha_f \rangle \rangle \rrbracket_{\rho, \alpha_f \mapsto \langle\langle \mathbf{int}, \mathbf{int}, R_f \rangle \rangle} = \llbracket \langle\langle \alpha_f \rangle \rangle \rrbracket_{\rho_1}$.

We now prove that $\langle\langle \langle f \rangle, V_1' \rangle, \langle\langle f \rangle, V_2' \rangle, \langle\langle \rangle, Q' \rangle \rangle \in \llbracket \Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_2}$.

We need to prove that:

- $\vdash_1 \langle\langle f \rangle, V_1' \rangle : \rho_{2L}(\Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P)$ and hence, $\vdash_1 \langle\langle f \rangle, V_1' \rangle : \Sigma\alpha_2 :$
 $\langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$ (since $\Sigma\alpha_2 : \langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$ is a closed signature)
- $\vdash_1 \langle\langle f \rangle, V_2' \rangle : \rho_{2R}(\Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P)$ and hence, $\vdash_1 \langle\langle f \rangle, V_2' \rangle : \Sigma\alpha_2 :$
 $\langle\langle \mathbf{int} \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P$
- $\langle\langle f \rangle, \langle\langle f \rangle, \langle\langle \rangle \rangle \rangle \in \llbracket \langle\langle \alpha_f \rightarrow \tau \rangle \rangle \rrbracket_{\rho_2}$
- $\langle V_1', V_2', Q' \rangle \in \llbracket \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_3}$, where $\rho_3 = \rho_2, \alpha_2 \mapsto \langle\langle \star, \star, \langle\langle \rangle \rangle \rangle$.

We have that:

- By using similar reasoning as in Case 2, $\vdash_1 \langle\langle f \rangle, V_1' \rangle : \rho_{2L}(\Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P)$
- Similarly, $\vdash_1 \langle\langle f \rangle, V_2' \rangle : \rho_{2R}(\Sigma\alpha_2 : \langle\langle \alpha_f \rightarrow \tau \rangle \rangle. \langle\langle L' \rangle \rangle_P)$
- We consider $\langle w_1, w_2 \rangle \in R_f$. From the definition of R_f , $\langle f w_1, f w_2 \rangle \in \llbracket \tau \rrbracket_\emptyset^{\text{ev}}$.
Thus, we have that $\langle f, f \rangle \in \llbracket \alpha_f \rightarrow \tau \rrbracket_{\rho_2}$. In other words, $\langle\langle f \rangle, \langle\langle f \rangle, \langle\langle \rangle \rangle \rangle \in$
 $\llbracket \langle\langle \alpha_f \rightarrow \tau \rangle \rangle \rrbracket_{\rho_2}$.
- We now prove $\langle V_1', V_2', Q' \rangle \in \llbracket \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_3}$. From IH, $\langle V_1', V_2', Q' \rangle \in \llbracket \langle\langle L' \rangle \rangle_P \rrbracket_{\rho'}$
Since $\langle \rho', \rho_3 \rangle \in \llbracket \cdot \rrbracket$ and $\vdash \langle\langle L' \rangle \rangle_P : \mathbf{sig}$ (Lemma 14), from Lemma 9, we have
that $\llbracket \langle\langle L' \rangle \rangle_P \rrbracket_{\rho'} = \llbracket \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_3}$. Thus, $\langle V_1', V_2', Q' \rangle \in \llbracket \langle\langle L' \rangle \rangle_P \rrbracket_{\rho_3}$.

H.2 Wrapper

Lemma 19. *If $\Gamma \vdash_P \langle e \rangle : \sigma$, then $\sigma = \langle \tau \rangle$ for some τ and $\Gamma \vdash e : \tau$.*

Proof. We prove this lemma by induction on the derivation of $\Gamma \vdash_P \langle e \rangle : \sigma$. We consider the last rule applied in the derivation. We have two cases (since other rules cannot be applied).

Case 1: Rule `ofm_dyn`.

$$\frac{\Gamma \vdash e : \tau}{\Gamma \vdash_P \langle e \rangle : \langle \tau \rangle}$$

The proof follows from the rule.

Case 2: Rule `ofm_subsume`.

$$\frac{\Gamma \vdash_P \langle e \rangle : \sigma' \quad \Gamma \vdash \sigma' \leq \sigma : \mathbf{sig}}{\Gamma \vdash_P \langle e \rangle : \sigma}$$

From the rule, we have that $\Gamma \vdash_P \langle e \rangle : \sigma'$. From IH, $\sigma' = \langle \tau' \rangle$ for some τ' and $\Gamma \vdash e : \tau'$. Since $\Gamma \vdash \sigma' \leq \sigma : \mathbf{sig}$, from Lemma 24, $\Gamma \vdash \sigma' \equiv \sigma : \mathbf{sig}$. From Lemma 23, it follows that $\sigma = \langle \tau \rangle$ for some τ s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbf{T}$. Thus, $\Gamma \vdash e : \tau$.

Lemma 12 (in §G). *If $\Gamma_P^{\mathcal{P}} \vdash e : \tau$, then $\Gamma_C^{\mathcal{P}} \vdash e$.*

Proof. First we have that $\vdash_P \lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle : \Pi^{\text{ap}} \alpha_P : \sigma_P. \langle \tau \rangle$.

$$\text{OFM_LAMAP} \frac{\vdash \sigma_P : \mathbf{sig} \quad \text{OFM_DYN} \frac{\alpha_P, m_P : \sigma_P \vdash e : \tau}{\alpha_P, m_P : \sigma_P \vdash_P \langle e \rangle : \langle \tau \rangle}}{\vdash_P \lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle : \Pi^{\text{ap}} \alpha_P : \sigma_P. \langle \tau \rangle}$$

From the weakening lemma (Lemma 7), we have that $\alpha_P, m_P : \sigma_P^C \vdash_P \lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle : \Pi^{\text{ap}} \alpha_P : \sigma_P. \langle \tau \rangle$. In addition, we have that $\alpha_P, m_P : \sigma_P^C \vdash \sigma_P^C \leq \sigma_P : \mathbf{sig}$ (Lemma 15 and Lemma 7) and $\alpha_P, m_P : \sigma_P^C \vdash \mathbf{Fst}(m_P) \gg \alpha_P$. Therefore, we have that:

$$\text{OFM_APPAP} \frac{\alpha_P, m_P : \sigma_P^C \vdash_P \lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle : \Pi^{\text{ap}} \alpha_P : \sigma_P. \langle \tau \rangle \quad \alpha_P, m_P : \sigma_P^C \vdash_P m : \sigma_P \quad \alpha_P, m_P : \sigma_P^C \vdash \mathbf{Fst}(m_P) \gg \alpha_P}{\alpha_P, m_P : \sigma_P^C \vdash_P (\lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle) m_P : \langle \tau \rangle [\alpha_P \mapsto \alpha_P]}$$

and hence, $\alpha_P, m_P : \sigma_P^C \vdash_P (\lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle) m_P : \langle \tau \rangle$.

Since $\alpha_P, m_P : \sigma_P^C \vdash \mathbf{Fst}(m_P) \gg \alpha_P$, from `mstep_app3`,

$$\alpha_P, m_P : \sigma_P^C \vdash (\lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle) m_P \rightarrow \alpha_P, m_P : \sigma_P^C \vdash e[m_P \mapsto m_P, \alpha_P \mapsto \alpha_P].$$

And thus,

$$\alpha_P, m_P : \sigma_P^C \vdash (\lambda^{\text{ap}} \alpha_P, m_P : \sigma_P. \langle e \rangle) m_P : \langle \tau \rangle \rightarrow \alpha_P, m_P : \sigma_P^C \vdash e.$$

From the type preservation theorem ([16, Theorem 2.2]), we have that $\alpha_P, m_P : \sigma_P^C \vdash_P \langle e \rangle : \langle \tau \rangle$. From Lemma 19, it follows that $\alpha_P, m_P : \sigma_P^C \vdash_P e$.

Lemma 20. *It follows that $\vdash_P V_P : \sigma_P^C$.*

Proof. We prove the lemma by induction on L .

Lemma 21. *It follows that $\vdash_I (V_P :> \sigma_P) : \sigma_P$.*

Proof. From Lemma 20, we have that $\vdash_P V_P : \sigma_P^C$. Since $\vdash \sigma_P^C \leq \sigma_P : \text{sig}$ (by Lemma 15), from the `ofm_subsume` rule, it follows that $\vdash_P V_P : \sigma_P$. From the `ofm_forget` rule, we have that $\vdash_I V_P : \sigma_P$. From the `ofm_seal` rule, we have that $\vdash_I (V_P :> \sigma_P) : \sigma_P$.

Lemma 22. *For any V , σ and σ' , if $\vdash_I (V :> \sigma) : \sigma'$ then $\vdash \sigma \leq \sigma' : \text{sig}$.*

Proof. We prove the lemma by induction on the derivation of $\vdash_I V :> \sigma : \sigma'$. We consider the last rule applied in the derivation. We have two cases.

Case 1: Rule `ofm_seal`. From the rule, we have that $\sigma = \sigma'$ and hence, $\vdash \sigma \leq \sigma' : \text{sig}$.

Case 2: Rule `ofm_subsume`. From the rule, we have $\vdash_I (V :> \sigma) : \sigma''$ and $\vdash \sigma'' \leq \sigma' : \text{sig}$. Since $\vdash_I (V :> \sigma) : \sigma''$, from IH, $\vdash \sigma \leq \sigma'' : \text{sig}$. Since $\vdash \sigma \leq \sigma'' : \text{sig}$ and $\vdash \sigma'' \leq \sigma' : \text{sig}$, from the `subs_trans` rule, it follows that $\vdash \sigma \leq \sigma' : \text{sig}$.

Lemma 23. *Suppose that $\Gamma \vdash \tau : \mathbb{T}$. It follows that:*

- if $\Gamma \vdash \sigma \equiv \langle \tau \rangle : \text{sig}$, then $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$,
- if $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$, then $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

Proof. We prove the lemma by induction on derivation of $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$ and $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$. We consider the last rule applied. We have four cases (other rules cannot be the last rule of the derivation).

Case 1: `eqs_refl`. From the rule, we have that $\sigma = \langle \tau \rangle$. Since $\Gamma \vdash \tau : \mathbb{T}$, from the `eqc_reflrule`, we have that $\Gamma \vdash \tau \equiv \tau : \mathbb{T}$.

Case 2: `eqs_symm`. We consider $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$ first. From the rule, we have that $\Gamma \vdash \sigma \equiv \langle \tau \rangle$. From IH, $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

We now consider $\Gamma \vdash \sigma \equiv \langle \tau \rangle : \text{sig}$. From the rule, we have that $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$. From IH, $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

Case 3: `eqs_trans`. We consider $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$ first. From the rule, we have that $\Gamma \vdash \langle \tau \rangle \equiv \sigma' : \text{sig}$ and $\Gamma \vdash \sigma' \equiv \sigma : \text{sig}$. From IH, it follows that $\sigma' = \langle \tau'' \rangle$ for some τ'' s.t. $\Gamma \vdash \tau \equiv \tau'' : \mathbb{T}$. From the static semantics, it follows that $\Gamma \vdash \tau'' : \mathbb{T}$. Thus, from IH, $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau'' \equiv \tau' : \mathbb{T}$. From the `eqc_trans` rule, it follows that $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

We now consider $\Gamma \vdash \sigma \equiv \langle \tau \rangle : \text{sig}$. From the rule, we have that $\Gamma \vdash \sigma \equiv \sigma' : \text{sig}$ and $\Gamma \vdash \sigma' \equiv \langle \tau \rangle : \text{sig}$. From IH, it follows that $\sigma' = \langle \tau'' \rangle$ for some τ'' s.t. $\Gamma \vdash \tau \equiv \tau'' : \mathbb{T}$. From the static semantics, it follows that $\Gamma \vdash \tau'' : \mathbb{T}$. Thus, from IH, $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau'' \equiv \tau' : \mathbb{T}$. From the `eqc_trans` rule, it follows that $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

Case 4: `eqs_dyn`. We consider $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$ first. From the rule, $\sigma = \langle \tau' \rangle$ for some τ' and $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

We now consider the case $\Gamma \vdash \sigma \equiv \langle \tau \rangle : \text{sig}$. From the rule, $\sigma = \langle \tau' \rangle$ for some τ' and $\Gamma \vdash \tau' \equiv \tau : \mathbb{T}$. From the `eqc_symm` rule, it follows that $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

Lemma 24. *Suppose that $\Gamma \vdash \langle \tau \rangle \leq \sigma : \text{sig}$. It follows that $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$.*

Proof. We prove this lemma by induction on the derivation of $\Gamma \vdash \langle \tau \rangle \leq \sigma : \text{sig}$. We consider the last rule applied. We have two cases (other rules cannot be the last rule in the derivation).

Case 1: *subs_refl*. From the rule, we have that $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$.

Case 2: *subs_trans*. From the rule, we have that $\Gamma \vdash \langle \tau \rangle \leq \sigma' : \text{sig}$ and $\Gamma \vdash \sigma' \leq \sigma : \text{sig}$. Since $\Gamma \vdash \langle \tau \rangle \leq \sigma' : \text{sig}$, from IH, we have that $\Gamma \vdash \langle \tau \rangle \equiv \sigma' : \text{sig}$. From Lemma 23, $\sigma' = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathsf{T}$. Since $\Gamma \vdash \sigma' \leq \sigma : \text{sig}$, we have that $\Gamma \vdash \langle \tau' \rangle \leq \sigma : \text{sig}$. From IH, we have that $\Gamma \vdash \langle \tau' \rangle \equiv \sigma : \text{sig}$. Since $\sigma' = \langle \tau' \rangle$, it follows that $\Gamma \vdash \sigma' \equiv \sigma : \text{sig}$.

Since $\Gamma \vdash \langle \tau \rangle \equiv \sigma' : \text{sig}$, and $\Gamma \vdash \sigma' \equiv \sigma : \text{sig}$, we have that $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \text{sig}$.

Lemma 25. *It follows that:*

- if $\Gamma \vdash \mathsf{T} \equiv k : \text{kind}$ then k is T ,
- if $\Gamma \vdash k \equiv \mathsf{T} : \text{kind}$ then k is T ,
- if $\Gamma \vdash 1 \equiv k : \text{kind}$ then k is 1 ,
- if $\Gamma \vdash k \equiv 1 : \text{kind}$ then k is 1 .

Proof. We prove this lemma by induction on the derivation of $\Gamma \vdash \mathsf{T} \equiv k : \text{kind}$, $\Gamma \vdash k \equiv \mathsf{T} : \text{kind}$, $\Gamma \vdash 1 \equiv k : \text{kind}$, and $\Gamma \vdash k \equiv 1 : \text{kind}$. We have three cases.

Case 1: Rule *eqk_refl*. We consider 1 first. From the rule, we have that k is 1 . The proof for T is similar.

Case 2: Rule *eqk_symm*.

- Case $\Gamma \vdash \mathsf{T} \equiv k : \text{kind}$. From the rule, $\Gamma \vdash k \equiv \mathsf{T} : \text{kind}$. From IH, k is T .
- Case $\Gamma \vdash k \equiv \mathsf{T} : \text{kind}$. From the rule, $\Gamma \vdash \mathsf{T} \equiv k : \text{kind}$. From IH, k is T .
- if $\Gamma \vdash 1 \equiv k : \text{kind}$. From the rule, $\Gamma \vdash k \equiv 1 : \text{kind}$. From IH, k is 1 .
- Case $\Gamma \vdash k \equiv 1 : \text{kind}$. From the rule, $\Gamma \vdash 1 \equiv k : \text{kind}$. From IH, k is 1 .

Case 3: Rule *eqk_trans*. The proof is similar to the proof of Case 2.

Lemma 26. *It follows that:*

- if $\Gamma \vdash 1 \equiv \sigma : \text{sig}$ then σ is 1 ,
- if $\Gamma \vdash \sigma \equiv 1 : \text{sig}$ then σ is 1 ,
- if $\Gamma \vdash \langle \mathsf{T} \rangle \equiv \sigma : \text{sig}$ then σ is $\langle \mathsf{T} \rangle$,
- if $\Gamma \vdash \sigma \equiv \langle \mathsf{T} \rangle : \text{sig}$ then σ is $\langle \mathsf{T} \rangle$.

Proof. We prove the two first items of the lemma by induction on the derivation of $\Gamma \vdash 1 \equiv \sigma : \text{sig}$ and $\Gamma \vdash \sigma \equiv 1 : \text{sig}$.

Case 1a: Rule *eqs_refl*. From the rule, σ is 1 .

Case 2a: Rule *eqs_symm*.

- Case $\Gamma \vdash 1 \equiv \sigma : \text{sig}$. From the rule, $\Gamma \vdash \sigma \equiv 1 : \text{sig}$. From IH, σ is 1 .
- Case $\Gamma \vdash \sigma \equiv 1 : \text{sig}$. From the rule, $\Gamma \vdash 1 \equiv \sigma : \text{sig}$. From IH, σ is 1 .

Case 3a: Rule eqs_trans. The proof is similar to the proof of Case 2a.

We prove the last two items of the lemma by induction on the derivation of $\Gamma \vdash \langle T \rangle \equiv \sigma : \text{sig}$, and $\Gamma \vdash \sigma \equiv \langle T \rangle : \text{sig}$.

Case 1b: Rule eqs_refl. From the rule, σ is $\langle T \rangle$.

Case 2b: Rule eqs_symm.

- Case $\Gamma \vdash \langle T \rangle \equiv \sigma : \text{sig}$. From the rule, $\Gamma \vdash \sigma \equiv \langle T \rangle : \text{sig}$. From IH, σ is $\langle T \rangle$.
- Case $\Gamma \vdash \sigma \equiv \langle T \rangle : \text{sig}$. From the rule, $\Gamma \vdash \langle T \rangle \equiv \sigma : \text{sig}$. From IH, σ is $\langle T \rangle$.

Case 3b: Rule eqs_trans. The proof is similar to the proof of Case 2b.

Case 4b: Rule eqs_stat. From the rule, $\sigma' = \langle k' \rangle$ for some k' s.t. $\Gamma \vdash T \equiv k' : \text{kind}$. From Lemma 25, k' is T . Thus, σ' is $\langle T \rangle$.

Lemma 27. *It follows that:*

- if $\Gamma \vdash 1 \leq k : \text{kind}$ then k is 1,
- if $\Gamma \vdash T \leq k : \text{kind}$ then k is T .

Proof. We prove this lemma by induction on the derivation of $\Gamma \vdash _ \leq k : \text{kind}$. Here, we only prove the first part of the lemma. The proof of the second part is similar.

Case 1: Rule subk_refl. From the rule, $\Gamma \vdash 1 \equiv k : \text{kind}$. From Lemma 25, k is 1.

Case 2: Rule subk_trans. From the rule, $\Gamma \vdash 1 \leq k' : \text{kind}$ and $\Gamma \vdash k' \leq k : \text{kind}$.

Since $\Gamma \vdash 1 \leq k' : \text{kind}$, from IH, k' is 1. Since $\Gamma \vdash k' \leq k : \text{kind}$ and k' is 1, from IH, we have that k is 1.

Lemma 28. *It follows that:*

- if $\Gamma \vdash 1 \leq \sigma : \text{sig}$ then σ is 1,
- if $\Gamma \vdash \langle T \rangle \leq \sigma : \text{sig}$ then σ is $\langle T \rangle$.

Proof. We prove the first part of the lemma by induction on the derivation of $\Gamma \vdash 1 \leq \sigma : \text{sig}$.

Case 1a: Rule subs_refl. From the rule, $\Gamma \vdash 1 \equiv \sigma : \text{sig}$. From Lemma 26, σ is 1.

Case 2a: Rule subs_trans. From the rule, $\Gamma \vdash 1 \leq \sigma' : \text{sig}$ and $\Gamma \vdash \sigma' \leq \sigma : \text{sig}$.

Since $\Gamma \vdash 1 \leq \sigma' : \text{sig}$, from IH, σ' is 1. Since $\Gamma \vdash \sigma' \leq \sigma : \text{sig}$ and σ' is 1, from IH, we have that σ is 1.

We now prove the second part of the lemma by induction on the derivation of $\Gamma \vdash \langle T \rangle \leq \sigma : \text{sig}$.

Case 1b: Rule subs_refl. From the rule, $\Gamma \vdash \langle T \rangle \equiv \sigma : \text{sig}$. From Lemma 26, σ is $\langle T \rangle$.

Case 2b: Rule subs_trans. From the rule, $\Gamma \vdash \langle T \rangle \leq \sigma' : \text{sig}$ and $\Gamma \vdash \sigma' \leq \sigma : \text{sig}$.

Since $\Gamma \vdash \langle \mathbb{T} \rangle \leq \sigma' : \mathbf{sig}$, from IH, σ' is $\langle \mathbb{T} \rangle$. Since $\Gamma \vdash \sigma' \leq \sigma : \mathbf{sig}$ and σ' is $\langle \mathbb{T} \rangle$, from IH, we have that σ is $\langle \mathbb{T} \rangle$.

Case 3b: Rule `subs_stat`. From the rule, σ is k' s.t. $\Gamma \vdash \mathbb{T} \leq k' : \mathbf{kind}$. From Lemma 27, $k' = \mathbb{T}$.

Lemma 29. *For any $L \subseteq \mathbf{V}_{\mathcal{P}}$, if $\Gamma \vdash \langle L \rangle_P \leq \sigma : \mathbf{sig}$ for some σ , then $\Gamma \vdash \langle L \rangle_P \equiv \sigma : \mathbf{sig}$.*

Proof. We prove the lemma by induction on L .

Case 1: $L = []$. We have that $\langle L \rangle_P = 1$. Therefore, we have that $\vdash 1 \leq \sigma : \mathbf{sig}$. From Lemma 28, we have that σ is 1 . Thus, $\Gamma \vdash 1 \equiv \sigma : \mathbf{sig}$.

Case 2: $L = x :: L'$ where $x \notin \text{dom}(\mathbf{F}_{\mathcal{P}})$. We have that $\langle L \rangle_P = \Sigma\alpha_x : \langle \mathbb{T} \rangle. \Sigma\alpha : \langle \alpha_x \rangle. \langle L' \rangle_P$. From the definition of subsignature, $\sigma = \Sigma\alpha_x : \sigma_1. \Sigma\alpha : \sigma_2. \sigma_3$. Without loss of generality, we suppose that α_x and α are not in $\text{dom}(\Gamma)$ (we can change the constructor variables if necessary). Therefore, we have that $\Gamma, \alpha_x : \mathbb{T} \text{ ok}$ and $\Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \text{ ok}$.

Since $\Gamma \vdash \langle L \rangle_P \leq \sigma : \mathbf{sig}$, from the `subs_sigma` rule, it follows that:

- $\Gamma \vdash \langle \mathbb{T} \rangle \leq \sigma_1 : \mathbf{sig}$
- $\Gamma, \alpha_x : \mathbb{T} \vdash \Sigma\alpha : \langle \alpha_x \rangle. \langle L' \rangle_P \leq \Sigma\alpha : \sigma_2. \sigma_3$, and hence,
 - $\Gamma, \alpha_x : \mathbb{T} \vdash \langle \alpha_x \rangle \leq \sigma_2 : \mathbf{sig}$
 - $\Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \vdash \langle L' \rangle_P \leq \sigma_3 : \mathbf{sig}$

We have that:

- $\Gamma \vdash \langle \mathbb{T} \rangle \leq \sigma_1 : \mathbf{sig}$. From Lemma 28, σ_1 is $\langle \mathbb{T} \rangle$ and hence, $\Gamma \vdash \langle \mathbb{T} \rangle \equiv \sigma_1 : \mathbf{sig}$.
- $\Gamma, \alpha_x : \mathbb{T} \vdash \langle \alpha_x \rangle \leq \sigma_2 : \mathbf{sig}$. From Lemma 24, $\Gamma, \alpha_x : \mathbb{T} \vdash \langle \alpha_x \rangle \equiv \sigma_2 : \mathbf{sig}$.
- $\Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \vdash \langle L' \rangle_P \leq \sigma_3 : \mathbf{sig}$. From IH, we have that $\Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \vdash \langle L' \rangle_P \equiv \sigma_3 : \mathbf{sig}$.

Therefore, we have the following derivation:

$$\text{EQS_SIGMA} \frac{\Gamma \vdash \langle \mathbb{T} \rangle \equiv \sigma_1 : \mathbf{sig} \quad \text{EQS_SIGMA} \frac{\Gamma, \alpha_x : \mathbb{T} \vdash \langle \alpha_x \rangle \equiv \sigma_2 : \mathbf{sig} \quad \Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \vdash \langle L' \rangle_P \equiv \sigma_3 : \mathbf{sig}}{\Gamma, \alpha_x : \mathbb{T} \vdash \Sigma\alpha : \langle \alpha_x \rangle. \langle L' \rangle_P \equiv \Sigma\alpha : \sigma_2. \sigma_3}}{\Gamma \vdash \Sigma\alpha_x : \langle \mathbb{T} \rangle. \Sigma\alpha : \langle \alpha_x \rangle. \langle L' \rangle_P \equiv \Sigma\alpha_x : \sigma_1. \Sigma\alpha : \sigma_2. \sigma_3}$$

Case 3: $= x :: L'$ and $\mathbf{F}_{\mathcal{P}}(x) = f$, where $\vdash f : \mathbf{int} \rightarrow \tau$. We have that $\langle L \rangle_P = \Sigma\alpha_f : \langle \mathbb{T} \rangle. \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P$. From the definition of subsignature, $\sigma = \Sigma\alpha_f : \sigma_1. \Sigma\alpha_1 : \sigma_2. \Sigma\alpha_2 : \sigma_3. \langle L' \rangle_P$. Without loss of generality, we suppose that α_f , α_1 , and α_2 are not in $\text{dom}(\Gamma)$ (we can change the constructor variables if it is necessary). Therefore, we have that $\Gamma, \alpha_f : \mathbb{T} \text{ ok}$, $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1 \text{ ok}$, and $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1, \alpha_2 : 1 \text{ ok}$.

Since $\Gamma \vdash \langle L \rangle_P \leq \sigma : \mathbf{sig}$, from the `subs_sigma` rule, it follows that:

- $\Gamma \vdash \langle \mathbb{T} \rangle \leq \sigma_1 : \mathbf{sig}$
- $\Gamma, \alpha_f : \mathbb{T} \vdash \Sigma\alpha_1 : \langle \alpha_f \rangle. \Sigma\alpha_2 : \langle \alpha_f \rightarrow \tau \rangle. \langle L' \rangle_P \leq \Sigma\alpha_1 : \sigma_2. \Sigma\alpha_2 : \sigma_3. \sigma_4$, and hence:
 - $\Gamma, \alpha_f : \mathbb{T} \vdash \langle \alpha_f \rangle \leq \sigma_2 : \mathbf{sig}$

- $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle\alpha_f \rightarrow \tau\rangle.\langle\langle L' \rangle\rangle_P \leq \Sigma\alpha_2 : \sigma_3.\sigma_4 : \mathbf{sig}$, and hence,
 - * $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1 \vdash \langle\alpha_f \rightarrow \tau\rangle \leq \sigma_3$
 - * $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1, \alpha_2 : 1 \vdash \langle\langle L' \rangle\rangle_P \leq \sigma_4 : \mathbf{sig}$

As in Case 2, we have that:

- $\Gamma, \alpha_f : \mathbb{T} \vdash \langle\alpha_f\rangle \equiv \sigma_2 : \mathbf{sig}$,
- $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1 \vdash \langle\alpha_f \rightarrow \tau\rangle \equiv \sigma_3$,
- $\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1, \alpha_2 : 1 \vdash \langle\langle L' \rangle\rangle_P \equiv \sigma_4 : \mathbf{sig}$,
- $\Gamma \vdash \langle\mathbb{T}\rangle \equiv \sigma_1 : \mathbf{sig}$,

Therefore, we have the following derivations:

$$\frac{\frac{\Gamma, \alpha_f : \mathbb{T} \vdash \langle\alpha_f\rangle \equiv \sigma_2 : \mathbf{sig} \quad \Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1 \vdash \langle\alpha_f \rightarrow \tau\rangle \equiv \sigma_3 \quad \Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1, \alpha_2 : 1 \vdash \langle\langle L' \rangle\rangle_P \equiv \sigma_4 : \mathbf{sig}}{\Gamma, \alpha_f : \mathbb{T}, \alpha_1 : 1 \vdash \Sigma\alpha_2 : \langle\alpha_f \rightarrow \tau\rangle.\langle\langle L' \rangle\rangle_P \equiv \Sigma\alpha_2 : \sigma_3.\sigma_4 : \mathbf{sig}}}{\Gamma, \alpha_f : \mathbb{T} \vdash \Sigma\alpha_1 : \langle\alpha_f\rangle.\Sigma\alpha_2 : \langle\alpha_f \rightarrow \tau\rangle.\langle\langle L' \rangle\rangle_P \equiv \Sigma\alpha_1 : \sigma_2.\Sigma\alpha_2 : \sigma_3.\sigma_4}$$

$$\frac{\frac{\Gamma \vdash \langle\mathbb{T}\rangle \equiv \sigma_1 : \mathbf{sig} \quad \Gamma, \alpha_f : \mathbb{T} \vdash \Sigma\alpha_1 : \langle\alpha_f\rangle.\Sigma\alpha_2 : \langle\alpha_f \rightarrow \tau\rangle.\langle\langle L' \rangle\rangle_P \equiv \Sigma\alpha_1 : \sigma_2.\Sigma\alpha_2 : \sigma_3.\sigma_4}{\Gamma \vdash \Sigma\alpha_f : \langle\mathbb{T}\rangle.\Sigma\alpha_1 : \langle\alpha_f\rangle.\Sigma\alpha_2 : \langle\alpha_f \rightarrow \tau\rangle.\langle\langle L' \rangle\rangle_P \equiv \Sigma\alpha_f : \sigma_1.\Sigma\alpha_1 : \sigma_2.\Sigma\alpha_2 : \sigma_3.\sigma_4}}$$

Thus, we have that $\Gamma \vdash \Sigma\alpha_f : \langle\mathbb{T}\rangle.\Sigma\alpha_1 : \langle\alpha_f\rangle.\Sigma\alpha_2 : \langle\alpha_f \rightarrow \tau\rangle.\langle\langle L' \rangle\rangle_P \equiv \Sigma\alpha_f : \sigma_1.\Sigma\alpha_1 : \sigma_2.\Sigma\alpha_2 : \sigma_3.\sigma_4$.

Lemma 30. *Let σ be a signature s.t. $\Gamma \vdash \sigma_P \equiv \sigma : \mathbf{sig}$. It follows that $\mathbf{Fst}(\sigma) = \mathbf{Fst}(\sigma_P)$.*

Proof. We claim that for any $L \subseteq \mathbf{V}_P$ and σ s.t. $\vdash \langle\langle L \rangle\rangle_P \equiv \sigma : \mathbf{sig}$, it follows that $\mathbf{Fst}(\sigma) = \mathbf{Fst}(\langle\langle L \rangle\rangle_P)$. The proof then follows directly from the claim. We prove the claim by induction on L .

Case 1: $L = []$. We have that $\langle\langle L \rangle\rangle_P = 1$. Since $\Gamma \vdash \sigma \equiv \sigma_P : \mathbf{sig}$, from Lemma 26, $\sigma = 1$. From the definition of $\mathbf{Fst}()$, we have that $\mathbf{Fst}(\sigma) = \mathbf{Fst}(\langle\langle L \rangle\rangle_P) = 1$.

Case 2: $L = x :: L'$, where $x \notin \text{dom}(\mathbf{F}_P)$. We have that $\langle\langle L \rangle\rangle_P = \Sigma\alpha_x : \langle\mathbb{T}\rangle.\Sigma\alpha : \langle\alpha_x\rangle.\langle\langle L' \rangle\rangle_P$. Since $\vdash \langle\langle L \rangle\rangle_P \equiv \sigma : \mathbf{sig}$, from the eqs_sigma rule, σ is $\Sigma\alpha_x : \sigma_1.\Sigma\alpha : \sigma_2.\sigma_3$ s.t.

- $\Gamma \vdash \langle\mathbb{T}\rangle \equiv \sigma_1 : \mathbf{sig}$
- $\Gamma, \alpha_x : \mathbb{T} \vdash \Sigma\alpha : \langle\alpha_x\rangle.\langle\langle L' \rangle\rangle_P \equiv \Sigma\alpha : \sigma_2.\sigma_3 : \mathbf{sig}$.
 - $\Gamma, \alpha_x : \mathbb{T} \vdash \langle\alpha_x\rangle \equiv \sigma_2 : \mathbf{sig}$
 - $\Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \vdash \langle\langle L' \rangle\rangle_P \equiv \sigma_3 : \mathbf{sig}$ (notice that since $\alpha_x : \mathbb{T}$, it follows that $\mathbf{Fst}(\langle\langle \tau \rangle\rangle) = 1$).

We have that:

- $\Gamma \vdash \langle\mathbb{T}\rangle \equiv \sigma_1 : \mathbf{sig}$. From Lemma 26, $\sigma_1 = \langle\mathbb{T}\rangle$. Thus, $\mathbf{Fst}(\sigma_1) = \mathbb{T} = \mathbf{Fst}(\langle\mathbb{T}\rangle)$.

- $\Gamma, \alpha_x : \mathbb{T} \vdash \langle \alpha_x \rangle \equiv \sigma_2 : \mathbf{sig}$. From Lemma 23, $\sigma_2 = \langle \tau \rangle$ for some τ s.t. $\Gamma, \alpha_x : \mathbb{T} \vdash \alpha_x \equiv \tau : \mathbb{T}$. Thus, $\mathbf{Fst}(\sigma_2) = 1 = \mathbf{Fst}(\langle \alpha_x \rangle)$
- $\Gamma, \alpha_x : \mathbb{T}, \alpha : 1 \vdash \langle \langle L' \rangle \rangle_P \equiv \sigma_3 : \mathbf{sig}$. From IH, we have that $\mathbf{Fst}(\langle \langle L' \rangle \rangle_P) = \mathbf{Fst}(\sigma_3)$.

Therefore, from the definition of $\mathbf{Fst}()$, we have that $\mathbf{Fst}(\langle \langle L \rangle \rangle_P) = \mathbf{Fst}(\sigma)$

Case 3: $L = x :: L'$, where $\mathbf{F}_P(x) = f$. The proof is similar to the proof of Case 2.

Lemma 31. – For any Γ , if $\Gamma \vdash \sigma \equiv \langle \tau \rangle : \mathbf{sig}$, then $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.
 – For any Γ , if $\Gamma \vdash \langle \tau \rangle \equiv \sigma : \mathbf{sig}$, then $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

Proof. We prove the lemma by induction on the derivation of the judgments. We only have the following cases, since other rules are not applicable.

Case 1: eqs_refl. The proof is trivial.

Case 2: eqs_symm. The proofs of two parts are directly from IH.

Case 3: eqs_trans. The proofs follows from IH and the fact that \equiv of types is transitivity.

Case 4: eqs_dyn. The proofs follows from the rule.

Lemma 32. – For any Γ , if $\Gamma \vdash \sigma \leq \langle \tau \rangle : \mathbf{sig}$, then $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.
 – For any Γ , if $\Gamma \vdash \langle \tau \rangle \leq \sigma : \mathbf{sig}$, then $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.

Proof. We prove both parts of the lemma by induction on the derivation of $\Gamma \vdash \sigma \leq \langle \tau \rangle : \mathbf{sig}$ and $\Gamma \vdash \langle \tau \rangle \leq \sigma : \mathbf{sig}$. We only have the following cases (other rules are not applicable).

Case 1: subs_refl. The proof follows from Lemma 31.

Case 2: subs_trans.

- Part 1: from the rule, $\Gamma \vdash \sigma \leq \sigma'' : \mathbf{sig}$ and $\Gamma \vdash \sigma'' \leq \langle \tau \rangle : \mathbf{sig}$. From IH for $\Gamma \vdash \sigma'' \leq \langle \tau \rangle : \mathbf{sig}$, $\sigma'' = \langle \tau'' \rangle$ for some τ'' s.t. $\Gamma \vdash \tau'' \equiv \tau' : \mathbb{T}$. Now, we can apply IH on $\Gamma \vdash \sigma \leq \langle \tau'' \rangle : \mathbf{sig}$ and we have that $\sigma = \langle \tau' \rangle$ for some τ' s.t. $\Gamma \vdash \tau' \equiv \tau'' : \mathbb{T}$. Since \equiv for type is transitivity, we have that $\Gamma \vdash \tau \equiv \tau' : \mathbb{T}$.
- Part 2: the proof is similar.

Lemma 33. If $\vdash_P \text{wrap}_P(e) : \Pi^{\text{gn}} \alpha_P : \sigma^*. \langle \tau \rangle$, where $\vdash \sigma^* \equiv \sigma_P : \mathbf{sig}$, then $\alpha_P / m_P : \sigma^{**} \vdash_I \langle e \rangle : \langle \tau' \rangle$ for some σ^{**} and τ' s.t. $\alpha_P / m_P : \sigma^{**} \vdash \tau \equiv \tau'$ and $\vdash \sigma^{**} \equiv \sigma_P$.

Proof. We prove the lemma by induction on the derivation of $\vdash_P \text{wrap}_P(e) : \Pi^{\text{gn}} \alpha_P : \sigma_P. \langle \tau \rangle$.

Case 1: ofm_lamgn. From the rule, $\alpha_P / m_P : \sigma^* \vdash_I \langle e \rangle : \langle \tau \rangle$.

Case 2: ofm_subsume. From the rule, we have $\vdash_P \text{wrap}_P(e) : \Pi^{\text{gn}} \alpha_P : \sigma. \sigma'$ s.t. $\vdash \Pi^{\text{gn}} \alpha_P : \sigma. \sigma' \leq \Pi^{\text{gn}} \alpha_P : \sigma^*. \langle \tau \rangle : \mathbf{sig}$. From subs_pigen, we have that $\vdash \sigma^* \leq \sigma : \mathbf{sig}$ and $\alpha_P : \mathbf{Fst}(\sigma^*) \vdash \sigma' \leq \langle \tau \rangle : \mathbf{sig}$.

- Since $\vdash \sigma^* \leq \sigma : \text{sig}$ and $\vdash \sigma^* \equiv \sigma_{\mathcal{P}} : \text{sig}$, we have that $\vdash \sigma_{\mathcal{P}} \leq \sigma : \text{sig}$. From Lemma 29, we have that $\vdash \sigma_{\mathcal{P}} \equiv \sigma : \text{sig}$.
- Since $\alpha_{\mathcal{P}} : \text{Fst}(\sigma^*) \vdash \sigma' \leq \langle \tau \rangle : \text{sig}$, from Lemma 32, we have that $\sigma' = \langle \tau' \rangle$ for some τ' .

Thus, we have that $\vdash_{\mathcal{P}} \text{wrap}_{\mathcal{P}}(e) : \Pi^{\text{gn}} \alpha_{\mathcal{P}} : \sigma. \langle \tau' \rangle$ s.t. $\vdash \sigma \equiv \sigma_{\mathcal{P}} : \text{sig}$. From IH, we close this case.

Lemma 34. *If $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_I \langle e \rangle : \langle \tau \rangle$ for some σ s.t. $\vdash \sigma \equiv \sigma_{\mathcal{P}} : \text{sig}$, then $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_{\mathcal{P}} \langle e \rangle : \langle \tau' \rangle$ for some τ' s.t. $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash \tau \equiv \tau' : \mathbb{T}$.*

Proof. We prove the lemma by induction on the derivation of $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_I \langle e \rangle : \langle \tau \rangle$. We only have the following case:

Case 1: `ofm_forget`. The proof is directly from the rule.

Case 2: `ofm_subsume`. From the rule, we have that $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_I \langle e \rangle : \sigma'$ and $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash \sigma' \leq \langle \tau \rangle : \text{sig}$. From Lemma 32, we have that $\sigma' = \langle \tau'' \rangle$ s.t. $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash \tau \equiv \tau'' : \mathbb{T}$. Thus, we can apply IH on $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_I \langle e \rangle : \sigma'$ and close this case.

Lemma 35. *If $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_{\mathcal{P}} \langle e \rangle : \langle \tau \rangle$ for some σ s.t. $\vdash \sigma \equiv \sigma_{\mathcal{P}} : \text{sig}$, then $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash e : \tau'$ for some τ' s.t. $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash \tau \equiv \tau' : \mathbb{T}$.*

Proof. We prove the lemma by induction on the derivation of $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash_{\mathcal{P}} \langle e \rangle : \langle \tau \rangle$. We have the following cases.

Case 1: `ofm_dyn`. The proof is directly from the rule.

Case 2: `ofm_subsume`. The proof is similar to the proof of Case 2 in Lemma 34.

Lemma 36. *If $\vdash \text{wrap}_{\mathcal{P}}(e) : \Pi^{\text{gn}} \alpha_{\mathcal{P}} : \sigma_{\mathcal{P}}. \langle \tau \rangle$, then $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash e : \tau'$ for some σ and τ' s.t. $\vdash \sigma \equiv \sigma_{\mathcal{P}} : \text{sig}$ and $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash \tau \equiv \tau' : \mathbb{T}$.*

Proof. Since $\vdash \text{wrap}_{\mathcal{P}}(e) : \Pi^{\text{gn}} \alpha_{\mathcal{P}} : \sigma_{\mathcal{P}}. \langle \tau \rangle$, from Lemma 33, we have that $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma'' \vdash_I \langle e \rangle : \langle \tau'' \rangle$ for some σ'' and τ'' s.t. $\vdash \sigma'' \equiv \sigma_{\mathcal{P}} : \text{sig}$ and $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma'' \vdash \tau \equiv \tau'' : \mathbb{T}$. From Lemma 34 and Lemma 35, we have that $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma' \vdash e : \tau'$ for some σ' and τ' s.t. $\vdash \sigma' \equiv \sigma_{\mathcal{P}} : \text{sig}$ and $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma' \vdash \tau \equiv \tau' : \mathbb{T}$.

Theorem 7 (in §G) If $\vdash_{\mathcal{P}} \text{wrap}_{\mathcal{P}}(e) : \Pi^{\text{gn}} \alpha_{\mathcal{P}} : \sigma_{\mathcal{P}}. \langle \tau \rangle$, then e is $\text{TRNI}(\mathcal{P}, \tau)$.

Proof. From Lemma 36, we have that $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash e : \tau'$ for some σ and τ' s.t. $\vdash \sigma \equiv \sigma_{\mathcal{P}} : \text{sig}$ and $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma \vdash \tau \equiv \tau' : \mathbb{T}$. From Lemma 30, we have that $\text{Fst}(\sigma) = \text{Fst}(\sigma_{\mathcal{P}})$. In addition, since module variables are not used in the judgments $\Gamma \vdash c : k$, we have that $\alpha_{\mathcal{P}} : \text{Fst}(\sigma_{\mathcal{P}}) \vdash \tau \equiv \tau' : \mathbb{T}$. Thus, we have that $\alpha_{\mathcal{P}}/m_{\mathcal{P}} : \sigma_{\mathcal{P}} \vdash e : \tau$. From Theorem 4, e is $\text{TRNI}(\mathcal{P}, \tau)$.

I Computation at different levels

To facilitate the presentation, in this section, we first present the encoding that support computation at different levels only for noninterference. Then we extend the encoding to support declassification policies.

I.1 Language and abstraction theorem

Language. To have an encoding that support multiple levels, we add the universally quantified types $\forall\alpha.\tau$ to the language presented in §2. In addition, to simplify the encoding, we add the unit type **unit**. W.r.t. these new types, we have new values: the unit value $\langle \rangle$ of **unit**, and values $\Lambda\alpha.e$ of type $\forall\alpha.\tau$. Therefore, in addition to the typing rules in §2, we have the following rules.

$$\frac{}{\Delta, \Gamma \vdash \langle \rangle : \mathbf{unit}} \qquad \frac{\Delta, \alpha, \Gamma \vdash e : \tau}{\Delta, \Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}$$

$$\frac{\Delta, \Gamma \vdash \tau' \quad \Delta, \Gamma \vdash \Lambda\alpha.e : \forall\alpha.\tau}{\Delta, \Gamma \vdash e[\tau'] : \tau[\tau'/\alpha]}$$

Similar to the language in §2, the semantics is a call-by-value semantic. We write $\lambda_- : \tau.e$ instead of $\lambda x : \tau.e$ when x is not a free variable in e .

Abstraction theorem. As in §F, an *environment* ρ is a mapping from type variables to tuples of the form $\langle \tau_1, \tau_2, R \rangle$ where $R \in \text{Rel}(\tau_1, \tau_2)$, and from term variables to tuples of the form $\langle v, v' \rangle$. For any ρ s.t. $\alpha \notin \text{dom}(\rho)$, we write $\rho, \alpha \mapsto \langle \tau_1, \tau_2, R \rangle$ to mean the environment ρ' s.t. $\text{dom}(\rho') = \text{dom}(\rho) \cup \{\alpha\}$ and $\rho'(\beta) = \langle \tau_1, \tau_2, R \rangle$ if $\beta = \alpha$, and $\rho'(\beta) = \rho(\beta)$ otherwise.

For the logical relation, in addition to the rules in Fig. 2, we have other rules described in Fig. 26. As in §2, we have Lemma 37 about the types of related values and terms are described.

$$\text{FR-UNIT} \frac{}{\langle \rangle, \langle \rangle \in [\mathbf{unit}]_\rho}$$

$$\text{FR-PAR} \frac{\forall R \in \text{Rel}(\tau_1, \tau_2). \langle v_1[\tau_1], v_2[\tau_2] \rangle \in [\tau]_{\rho, \alpha \mapsto \langle \tau_1, \tau_2, R \rangle}^{\text{ev}}}{\langle v_1, v_2 \rangle \in [\forall\alpha.\tau]_\rho}$$

Fig. 26. Logical relation

Lemma 37. *We have that:*

- if $\langle v_1, v_2 \rangle \in [\tau]_\rho$, then $\vdash v_1 : \rho_L(\tau)$, $\vdash v_2 : \rho_R(\tau)$ and
- if $\langle e_1, e_2 \rangle \in [\tau]_\rho^{\text{ev}}$, then $\vdash e_1 : \rho_L(\tau)$ and $\vdash e_2 : \rho_R(\tau)$.

We write $\rho \models \Delta$ to mean that the domain of ρ is Δ (i.e. $\text{dom}(\rho) = \Delta$). Let $\rho|_{tv}$ be the restriction of ρ to type variables. We write $\rho \models \Delta, \Gamma$ to mean that $\rho|_{tv} \models \Delta$ and

- the set of term variables in $\text{dom}(\rho)$ is equal to $\text{dom}(\Gamma)$ (i.e. $\text{dom}(\Gamma) = \{x \mid x \in \text{dom}(\rho)\}$),

- for all $x \in \text{dom}(\rho)$, $\rho(x) \in \llbracket \Gamma(x) \rrbracket_\rho$.

We define $\Delta, \Gamma \vdash e_1 \sim e_2 : \tau$ as in §2 and §F. The following theorem says that the logical equivalence is reflexive.

Theorem 8 (Abstraction Theorem). *If $\Delta, \Gamma \vdash e : \tau$, then $\Delta, \Gamma \vdash e \sim e : \tau$.*

I.2 Noninterference for free

Overview We consider noninterference defined for an arbitrary finite security lattice $\langle \mathcal{L}, \sqsubseteq \rangle$, where \mathcal{L} is the set of security levels, and \sqsubseteq is the order between them. We use lvl as a function from variables to security levels.

We consider programs that read input values from input channels and generate output values to output channels. Before presenting assumptions on inputs, outputs and programs, we introduce some auxiliary notations. Let $\{W_l\}_{l \in \mathcal{L}}$ be a family of type constructors indexed by l . A *wrapped value* of type τ at level l is a value of type $W_l \tau$. A wrapped value v can be unwrapped by $\text{unwrap } k \ v$, where k is an appropriate key and $\text{unwrap } k \ v$ can be implemented as $v \ k$. Below are assumptions on input/output channels and programs.

- Assumption 9**
1. Each input/output channel is associated with a security level.
 2. The type of an input value from an input channel is **int**. The type of an output value to an output channel is **int**.¹⁷
 3. Free variables in programs are considered as inputs from input channels.
 4. Input values on an input channel at level l cannot be directly observed. We model this by using $W_l \text{int}$ as their type, not **int**.
 5. Programs are executed in a context where there are several output channels, each corresponding to a security level. A program will compute a tuple of wrapped output values, where each element of the tuple is unwrapped by using unwrap and an appropriate key, and the unwrapped value is sent to a channel. The assumption about execution of programs in the context is illustrated in the following pseudo program, where e is a program satisfying assumptions, o is the computed tuple, Output.Channel_l is an output channel at l , k_l is a key to unwrap value at l , pr_l projects the output value for the output channel l .

```

let o = e in
  IO.Channell1 := unwrap kl1 (πl1 o)
  ⋮
  IO.Channelln := unwrap kln (πln o)

```

Therefore, outputs of considered programs are of the type $W_{l_1} \text{int} \times \dots \times W_{l_n} \text{int}$.

¹⁷ In this section, the type of inputs/outputs is **int**. We choose this since we are sticking with [27]. The result presented in this paper can be generalized to accepting confidential inputs of arbitrary types (e.g. **Bool**, **String**, etc).

Encoding For every level l in the lattice, we introduce a type variable α_l . To protect data of type τ at level $l \in \mathcal{L}$, we use a type variable α_l and form the type $\alpha_l \rightarrow \tau$. In other words, $\mathbf{W}_l \mathbf{int}$ is $\alpha_l \rightarrow \tau$. Therefore, an input of type \mathbf{int} at level l is encoded as a value of type $\alpha_l \rightarrow \mathbf{int}$.

To do computation on values of types $\alpha_l \rightarrow \tau$, programmers have to manage key manually and it is error-prone. To support handling values of $\alpha_l \rightarrow \tau$, we propose the following interfaces that satisfy monad laws [53,33] (see proof in §I.2). The detailed implementation of them will be described later.

- \mathbf{cp}_l : this interface transfers a wrapped value at l to a continuation and produce an output at l ,
- \mathbf{wr}_l : this interface is used to wraps terms.

In addition, we have the interface $\mathbf{cvu}_l^{l'}$ (for $l \sqsubseteq l'$) which is used to convert data from a level l to a higher level l' . Therefore, we have the typing context Δ_{NI} and the term context Γ_{NI} for NI as described in Fig. 27. Examples illustrating \mathbf{cp}_l and $\mathbf{cv}_l^{l'}$ are in Example 22 and Example 23.¹⁸

$$\begin{aligned}
\Delta_{NI} &= \{\alpha_l \mid l \in \mathcal{L}\} \\
\Gamma_{NI} &= \{x : \alpha_l \rightarrow \mathbf{int} \mid x \text{ is an input} \wedge \text{lvl}(x) = l\} \cup \\
&\quad \{\mathbf{cp}_l : \forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2 \mid l \in \mathcal{L}\} \cup \\
&\quad \{\mathbf{cvu}_l^{l'} : \forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta) \mid l, l' \in \mathcal{L} \wedge l \sqsubseteq l'\} \cup \\
&\quad \{\mathbf{wr}_l : \forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta \mid l \in \mathcal{L}\}
\end{aligned}$$

Fig. 27. Contexts for noninterference

Example 22 (Computation at a level). We consider the lattice $\langle \mathcal{L}_3, \sqsubseteq \rangle$ where $\mathcal{L}_3 = \{L, M, H\}$ and $L \sqsubseteq M \sqsubseteq H$. Suppose that there are three inputs hi , mi and li and their associated levels are respectively H , M , and L . By replacing \mathcal{L} in Fig. 27 with \mathcal{L}_3 , we have the typing context Δ_{NI} and the term context Γ_{NI} for NI defined on $\langle \mathcal{L}_3, \sqsubseteq \rangle$. Note that the types of hi , mi and li in Γ_{NI} are respectively $\alpha_H \rightarrow \mathbf{int}$, $\alpha_M \rightarrow \mathbf{int}$, and $\alpha_L \rightarrow \mathbf{int}$.

We now illustrate \mathbf{cp} by having a program that has a computation at M : we want to have $mi + 1$.¹⁹

In order to use \mathbf{cp}_M , we first wrap $\mathit{plus_one}$ as below.

$$\mathit{wrap_plus_one} = \lambda x : \mathbf{int}. \mathbf{wr}_M(\mathit{plus_one} \ x)$$

¹⁸ In these two examples, for illustration purpose, programs only generate single inputs at single levels. A program generating multiple outputs for multiple levels is illustrated later in Example 25.

¹⁹ Input mi is not of the \mathbf{int} type and hence, $mi + 1$ is not well-typed. We use it just to express the idea of the computation. This convention is also used in other examples.

Then we have the following program e_1 of the type $\alpha_M \rightarrow \mathbf{int}$ encoding the requirement that the function *plus_one* is applied on y and the result is at M .

$$e_1 = \mathbf{cp}_M[\mathbf{int}][\mathbf{int}] \text{ } mi \text{ } wrap_plus_one$$

Example 23 (Convert to a higher level). In this example, we will do the computation $hi + mi$ and the result is at H . This example illustrates the usage of $\mathbf{cv}_{l'}$ (for $l \sqsubseteq l'$).

Let *add* be a function of the type $\mathbf{int} \rightarrow \mathbf{int} \rightarrow \mathbf{int}$. From *add*, we construct *wrap_addc* of the type $\mathbf{int} \rightarrow \alpha_H \rightarrow \mathbf{int}$, where c is a variable of the type \mathbf{int} .

$$wrap_addc = \lambda y : \mathbf{int}. \mathbf{wr}_H(add \text{ } c \text{ } y)$$

The following program e_2 of the type $\alpha_H \rightarrow \mathbf{int}$ computes the sum of hi and mi and the computed sum is at H . Note that in order to transfer mi at M to the continuation *wrap_addc* by using \mathbf{cp}_H , we need to convert mi from level M to level H by using \mathbf{cv}_M^H .

$$e_2 = \mathbf{cp}_H[\mathbf{int}][\mathbf{int}] \text{ } hi \text{ } (\lambda c : \mathbf{int}. (\mathbf{cp}_H[\mathbf{int}][\mathbf{int}] (\mathbf{cv}_M^H mi) wrap_addc))$$

Indistinguishability In this section, we define indistinguishability for an observer $\zeta \in \mathcal{L}$.²⁰ As mentioned in §I.2, α_l is used to protect data at l . We can think of closed terms related at α_l as keys to open wrapped data at l . Thus, if an observer ζ can observe protected data at l (i.e. $l \sqsubseteq \zeta$), this observer has the keys. Otherwise, the observer has no key. This idea is captured in the following definition, where we use **unit** as the type of keys,²¹ and $\mathbf{Full}_{\mathbf{unit}} = \{\langle \rangle, \langle \rangle\}$.²²

Definition 10. *An environment ρ is an environment for noninterference w.r.t. an observer ζ (denoted by $\rho \models_{\zeta} \mathbf{NI}$) if $\rho \models \Delta_{\mathbf{NI}}$ and for any $\alpha_l \in \Delta_{\mathbf{NI}}$:*

$$\rho(\alpha_l) = \begin{cases} \langle \mathbf{unit}, \mathbf{unit}, \mathbf{Full}_{\mathbf{unit}} \rangle & \text{if } l \sqsubseteq \zeta, \\ \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle & \text{if } l \not\sqsubseteq \zeta. \end{cases}$$

Note that there is only a unique an environment for NI w.r.t. an observer ζ and hence, hereafter we will use the environment for NI w.r.t. an observer. We next define indistinguishability for an observer ζ as an instantiation of the logical relation.

Definition 11. *Given a type τ s.t. $\Delta_{\mathbf{NI}} \vdash \tau$. The indistinguishability relations on values and terms of τ for an observer ζ (denoted by resp. $I_{\mathbf{NI}}^{\zeta}[\tau]$ and $I_{\mathbf{NI}}^{\zeta}[\tau]^{ev}$) are defined as*

$$I_{\mathbf{NI}}^{\zeta}[\tau] = \llbracket \tau \rrbracket_{\rho} \quad I_{\mathbf{NI}}^{\zeta}[\tau]^{ev} = \llbracket \tau \rrbracket_{\rho}^{ev}$$

where ρ is the environment for NI w.r.t the observer ζ .

²⁰ Following [12], we use ζ for observers.

²¹ Note that other types can be used.

²² When another closed type τ is used as the type of keys, we require that all keys are indistinguishable and hence, the substitution for α_l is \mathbf{Full}_{τ} when $l \sqsubseteq \zeta$. To emphasize this, in Def. 10, we use $\mathbf{Full}_{\mathbf{unit}}$ instead of the identity relation on **unit**, even though for **unit**, these two relations coincide.

Example 24 (Indistinguishability). We consider the lattice $\langle \mathcal{L}_3, \sqsubseteq \rangle$ described in Example 22. We will describe the indistinguishability relations of the type $\alpha_M \rightarrow \mathbf{int}$ for an observer at L and for an observer at H . Note that $\alpha_M \rightarrow \mathbf{int}$ is the type of inputs at level M .

The intuition is that the observer at H can observe data at M and hence, two wrapped values at M are indistinguishable to the observer at H if the wrapped values are the same. This intuition is captured by the definition of indistinguishability as demonstrated below, where ρ is an environment s.t. $\rho \models_H \text{NI}$.

$$\begin{aligned}
I_{NI}^H[\alpha_M \rightarrow \mathbf{int}] &= \quad (\text{from Def. 11, Lem. 1, and rule FR-Fun}) \\
&\quad \{ \langle \lambda x : \mathbf{unit}.e_1, \lambda x : \mathbf{unit}.e_2 \rangle \mid \vdash \lambda x : \mathbf{unit}.e_i : \mathbf{unit} \rightarrow \mathbf{int}, \\
&\quad \quad \forall (v_1, v_2) \in \llbracket \alpha_M \rrbracket_\rho. ((\lambda x : \mathbf{unit}.e_1) v_1, (\lambda x : \mathbf{unit}.e_2) v_2) \in \llbracket \mathbf{int} \rrbracket_\rho^{\text{ev}} \} \\
&= \quad (\text{from Def. 10, rule FR-Term, rule FR-Int}) \\
&\quad \{ \langle \lambda x : \mathbf{unit}.e_1, \lambda x : \mathbf{unit}.e_2 \rangle \mid \vdash \lambda x : \mathbf{unit}.e_i : \mathbf{unit} \rightarrow \mathbf{int}, \\
&\quad \quad \exists n_1, n_2. \quad e_1[\langle \rangle / x] \rightarrow^* n_1, \quad e_2[\langle \rangle / x] \rightarrow^* n_2 \}
\end{aligned}$$

Therefore, for any n_1 and n_2 , $(\lambda_- : \mathbf{unit}.n_1, \lambda_- : \mathbf{unit}.n_2) \in I_{NI}^H[\alpha_M \rightarrow \mathbf{int}]$ iff $n_1 = n_2$.

While the observer at H can observe data at M , the observer at L cannot. Therefore, to the observer at L , all wrapped values at M are indistinguishable. Below is a demonstration showing that indistinguishability for the observer at L is as desired, where $\rho \models_L \text{NI}$.

$$\begin{aligned}
I_{NI}^L[\alpha_M \rightarrow \mathbf{int}] &= \quad (\text{from Def. 11 and rule FR-Fun}) \\
&\quad \{ \langle \lambda x : \mathbf{unit}.e_1, \lambda x : \mathbf{unit}.e_2 \rangle \mid \vdash \lambda x : \mathbf{unit}.e_i : \mathbf{unit} \rightarrow \mathbf{int}, \\
&\quad \quad \forall (v_1, v_2) \in \llbracket \alpha_M \rrbracket_\rho. ((\lambda x : \mathbf{unit}.e_1) v_1, (\lambda x : \mathbf{unit}.e_2) v_2) \in \llbracket \mathbf{int} \rrbracket_\rho^{\text{ev}} \} \\
&= \quad (\text{since } I_{NI}^L[\alpha_M] = \emptyset, \text{ the condition on } v_1 \text{ and } v_2 \text{ holds vacuously}) \\
&\quad \{ \langle \lambda x : \mathbf{unit}.e_1, \lambda x : \mathbf{unit}.e_2 \rangle \mid \vdash \lambda x : \mathbf{unit}.e_i : \mathbf{unit} \rightarrow \mathbf{int} \}
\end{aligned}$$

Therefore, for any n_1 and n_2 , $(\lambda_- : \mathbf{unit}.n_1, \lambda_- : \mathbf{unit}.n_2) \in I_{NI}^H[\alpha_M \rightarrow \mathbf{int}]$.

Typing implies non-interference

Interface for computation. The implementations of cp_l , $\text{cv}_{ll'}$, and wr_l are respectively **comp**, **convup**, and **wrap** as below. The construction of **convup** and **wrap** is straightforward. On a protected input of type $\mathbf{unit} \rightarrow \beta_1$ and a continuation of type $\beta_1 \rightarrow (\mathbf{unit} \rightarrow \beta_2)$, **comp** first unfolds the protected input by applying it to the key $\langle \rangle$ and then applies the continuation on the result.

$$\begin{aligned}
\text{comp} &= \Lambda \beta_1, \beta_2. \lambda x : \mathbf{unit} \rightarrow \beta_1. \lambda f : \beta_1 \rightarrow (\mathbf{unit} \rightarrow \beta_2). f(x \langle \rangle) \\
\text{convup} &= \Lambda \beta. \lambda x : \mathbf{unit} \rightarrow \beta. x \\
\text{wrap} &= \Lambda \beta. \lambda x : \beta. \lambda_- : \mathbf{unit}. x
\end{aligned}$$

We next define a full environment for NI which covers not only type variables but also term variables. Basically, its restriction to type variables (denoted by $\rho|_{tv}$) is the environment for NI w.r.t. an observer ζ . In addition, it maps inputs to indistinguishable values and interfaces for computation to the specific implementations just defined.

Definition 12. *An environment ρ is a full environment for NI w.r.t. an observer ζ (denoted by $\rho \models_{\zeta}^{full} NI$) if $\rho|_{tv} \models_{\zeta} NI$ and*

- for all x of the type $\alpha_l \rightarrow \mathbf{int}$ (i.e. different from \mathbf{cp}_l and $\mathbf{cvu}_l^{l'}$),

$$\rho(x) = \langle v_1, v_2 \rangle \in I_{NI}^{\zeta}[\alpha_l \rightarrow \mathbf{int}],$$

- for all l , $\rho(\mathbf{cp}_l) = \langle \mathbf{comp}, \mathbf{comp} \rangle$,
- for all l and l' s.t. $l \sqsubset l'$, $\rho(\mathbf{cvu}_l^{l'}) = \langle \mathbf{convup}, \mathbf{convup} \rangle$
- for all l , $\rho(\mathbf{wr}_l) = \langle \mathbf{wrap}, \mathbf{wrap} \rangle$.

NI for free. In order to instantiate the abstraction theorem to prove that a program is NI, we need to prove that **comp** is indistinguishable to itself for any observer and so are **convup** and **wrap**.²³

Lemma 38. *For any ζ , it follows that:*

$$\begin{aligned} \langle \mathbf{comp}, \mathbf{comp} \rangle &\in I_{NI}^{\zeta}[\forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2] \\ \langle \mathbf{convup}, \mathbf{convup} \rangle &\in I_{NI}^{\zeta}[\forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta)] \\ \langle \mathbf{wrap}, \mathbf{wrap} \rangle &\in I_{NI}^{\zeta}[\forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta]. \end{aligned}$$

From the definition of $\rho \models_{\zeta}^{full} NI$ and Lemma 38, we have the following corollary.

Corollary 1. *For any $\rho \models_{\zeta}^{full} NI$, $\rho \models \Delta_{NI}, \Gamma_{NI}$.*

We next prove that if a program is well-typed in Δ_{NI}, Γ_{NI} , then it transforms indistinguishable inputs to indistinguishable outputs.

Theorem 10. *If $\Delta_{NI}, \Gamma_{NI} \vdash e : \tau$, then for any $\zeta \in \mathcal{L}$ and $\rho \models_{\zeta}^{full} NI$,*

$$\langle \rho_L(e), \rho_R(e) \rangle \in I_{NI}^{\zeta}[\tau]^{ev}.$$

Proof. As proven in Corollary 1, we have that $\rho \models \Delta_{NI}, \Gamma_{NI}$. Since $\Delta_{NI}, \Gamma_{NI} \vdash e : \tau$, from Theorem 8, we have that $\langle \rho_L(e), \rho_R(e) \rangle \in \llbracket \tau \rrbracket_{\rho}^{ev}$. From the definition of indistinguishability, it follows that $\langle \rho_L(e), \rho_R(e) \rangle \in I_{NI}^{\zeta}[\tau]^{ev}$.

Therefore, we have the following corollary for programs of the type $(\alpha_{l_1} \rightarrow \mathbf{int}) \times \dots \times (\alpha_{l_n} \rightarrow \mathbf{int})$.

²³ Note that Lemma 38 is not a direct result of the abstraction theorem since the types of implementations (e.g. **comp**) of defined interfaces are closed types while the types of interfaces (e.g. the type of \mathbf{cp}_l) are open types.

Corollary 2 (NI for free). *If $\Delta_{NI}, \Gamma_{NI} \vdash e : (\alpha_{l_1} \rightarrow \mathbf{int}) \times \cdots \times (\alpha_{l_n} \rightarrow \mathbf{int})$, then for any $\zeta \in \mathcal{L}$ and $\rho \models_{\zeta}^{full} NI$,*

$$\langle \rho_L(e), \rho_R(e) \rangle \in \check{I}_{NI}[(\alpha_{l_1} \rightarrow \mathbf{int}) \times \cdots \times (\alpha_{l_n} \rightarrow \mathbf{int})]^{ev}.$$

Example 25. By reusing programs in Example 22 and Example 23, we here present of a program that generates output values for levels H , M , and L : the output of the program in this example is of the type $(\alpha_H \rightarrow \mathbf{int}) \times (\alpha_M \rightarrow \mathbf{int}) \times (\alpha_L \rightarrow \mathbf{int})$.

To make the paper easier to follow, we recall here e_1 and e_2 in resp. Example 22 and Example 23. Programs e_1 and e_2 compute resp. $mi+1$ and $hi+mi+1$.

$$\begin{aligned} e_1 &= \mathbf{cp}_M[\mathbf{int}][\mathbf{int}] \text{ } mi \text{ } \mathbf{wrap_plus_one} \\ e_2 &= \mathbf{cp}_H[\mathbf{int}][\mathbf{int}] \text{ } hi \text{ } (\lambda c : \mathbf{int}. (\mathbf{cp}_H[\mathbf{int}][\mathbf{int}] \text{ } (\mathbf{cvu}_M^H \text{ } mi) \text{ } \mathbf{wrap_adde})) \end{aligned}$$

Below is the program e that computes output values for all levels. From Corollary 2, this program is NI.

$$e = (\lambda mi : \alpha_M \rightarrow \mathbf{int}. \langle e_2, \langle mi, li \rangle \rangle) e_1$$

Monadic encoding As mentioned earlier, our encoding is a monadic encoding. We first recall here the monad laws in the popular infix form [53], where **wrap** and **comp** are respectively the unit and the bind expressions in monad. Note that we write $e \cong e'$ when $\langle e, e' \rangle \in \llbracket \tau \rrbracket_{\emptyset}^{ev}$ which intuitively means that both e and e' reduce to an equal value. To simplify the presentation, we do not include types in these laws (e.g. instead of writing $\mathbf{wrap}[\tau] \text{ } e$, we just write $\mathbf{wrap} \text{ } e$).

$$\begin{aligned} (\mathbf{wrap} \text{ } e) \text{ } \mathbf{comp} \text{ } f &\cong f \text{ } e && \text{Left unit} \\ e \text{ } \mathbf{comp} \text{ } \mathbf{wrap} &\cong e && \text{Right unit} \\ (e \text{ } \mathbf{comp} \text{ } f) \text{ } \mathbf{comp} \text{ } g &\cong e \text{ } \mathbf{comp} \text{ } (\lambda x. ((f \text{ } x) \text{ } \mathbf{comp} \text{ } g)) && \text{Associativity} \end{aligned}$$

Next we prove that our encoding satisfies monad laws in Lemma 39. We do not use the infix notation in the lemma. The first, the second and the third parts of the lemma correspond to resp. the left unit, right unit and associativity laws.

Lemma 39. *For any closed types τ and τ' :*

1. *for any e s.t. $\vdash e : \tau$, for any $f : \tau \rightarrow \mathbf{unit} \rightarrow \tau'$,*

$$\mathbf{comp}[\tau][\tau'] (\mathbf{wrap}[\tau] \text{ } e) \text{ } f \cong f \text{ } e$$

2. *for any e s.t. $\vdash e : \mathbf{unit} \rightarrow \tau$,*

$$\mathbf{comp}[\tau][\tau'] \text{ } e \text{ } \mathbf{wrap}[\tau] \cong e$$

3. *for any e s.t. $\vdash e : \mathbf{unit} \rightarrow \tau$, $\vdash f : \tau \rightarrow \mathbf{unit} \rightarrow \tau''$, $\vdash g : \tau'' \rightarrow \mathbf{unit} \rightarrow \tau$*

$$\mathbf{comp}[\tau''][\tau'] (\mathbf{comp}[\tau][\tau''] \text{ } e \text{ } f) \text{ } g \cong \mathbf{comp}[\tau][\tau'] \text{ } e \text{ } (\lambda x : \tau. \mathbf{comp}[\tau''][\tau'] (f \text{ } x) \text{ } g)$$

I.3 Type-based relaxed noninterference for free

Declassification policies Declassification policies considered in this section are similar to the ones in §3, except that since we have multiple levels, for an input at l that can be declassified via f , we need to specify which level it can be declassified to. Therefore, we define declassification policies as below, where **Dec** is the set of all closed declassification functions of types $\mathbf{int} \rightarrow \tau_f$ for some closed τ_f , and \hookrightarrow is used to denote partial functions.

Definition 13 (Declassification policies). *A declassification policy \mathcal{P} is a tuple $\langle \langle \mathcal{L}, \sqsubseteq \rangle, \mathbf{I}, \text{lvl}, \mathbf{F} \rangle$, where $\langle \mathcal{L}, \sqsubseteq \rangle$ is a finite lattice of security levels, \mathbf{I} is a finite set of inputs, $\text{lvl} : \mathbf{I} \rightarrow \mathcal{L}$ is a mapping from inputs to security levels, $\mathbf{F} : \mathbf{I} \hookrightarrow (\mathbf{Dec} \times \mathcal{L})$ is a partial function from inputs to declassification functions and security levels s.t. if $\mathbf{F}(x) = \langle f, l \rangle$ then $\text{lvl}(x) \not\sqsubseteq l$.*

Example 26 (Policy \mathcal{P}_{MOE}). Consider policy \mathcal{P}_{MOE} given by $\langle \langle \mathcal{L}_3, \sqsubseteq \rangle, \mathbf{I}, \text{lvl}, \mathbf{F} \rangle$, where

- $\langle \mathcal{L}_3, \sqsubseteq \rangle$ is the lattice with three levels L , M , and H as described in Example 11,
- $\mathbf{I} = \{hi, mi, li\}$: there are three inputs: hi , mi and li ,
- associated levels of inputs are respectively H , M , and L ,
- input mi can be declassified via $f = \lambda x : \mathbf{int}. x \bmod 2$ to level L (i.e. $\mathbf{F}(mi) = (f, L)$).

Encoding In this section, we extend the encoding and the indistinguishability relation for NI presented in §I.2 to counterparts for declassification policies. Through this section, we consider a fixed policy \mathcal{P} given by $\langle \langle \mathcal{L}, \sqsubseteq \rangle, \mathbf{I}, \text{lvl}, \mathbf{F} \rangle$.

In addition to assumptions in §I.2, for declassification policies, we additionally assume that declassifiers are also inputs of programs: that is for each input x that can be declassified via f , we have an input x_f which is corresponding for f .

As in NI, we have type variables α_l for all levels $l \in \mathcal{L}$, and we also have interfaces \mathbf{cp}_l , \mathbf{cvu}_l , and \mathbf{wr}_l . The encoding for inputs that cannot be declassified (i.e. inputs that are not in the domain of \mathbf{F}) is the same as the encoding for inputs in NI: the type of an input x at level l is $\mathbf{W}_l \mathbf{int} = \alpha_l \rightarrow \mathbf{int}$.

We consider an input x at level l that can be declassified via f of the type $\mathbf{int} \rightarrow \tau_f$ to level l' where $l' \sqsubseteq l$. As for NI, we may use $\alpha_l \rightarrow \mathbf{int}$ as the type for this input and require that only observers at l or higher levels have the key to unwrap values at l . However, as explained in Example 24, to an observer at l' , all wrapped values at l are indistinguishable and we do not want this. Therefore, we introduce a fresh type variable α_l^f for keys of this input and we require that the observer at l' also has keys. Since the observer at l' has key, if we use $\alpha_l^f \rightarrow \mathbf{int}$ as the type for the input, as explained in Example 24, only equal values are indistinguishable. Therefore, we use $\alpha_l^f \rightarrow \alpha^f$ as the type of the input, where α_f is a fresh variable.

We now look at the input x_f which corresponds to the declassifier f for x . Following the idea of the monadic encoding, we can use $\alpha^f \rightarrow \alpha_{l'} \rightarrow \tau_f$ as the type for x_f and we will define an interface cpdec_x as below to apply x_f to x .

$$\text{cpdec}_x : (\alpha_l^f \rightarrow \alpha^f) \rightarrow (\alpha_f \rightarrow \alpha_{l'} \rightarrow \tau_f) \rightarrow \alpha_{l'} \rightarrow \tau_f$$

This interface can be used to apply different functions of type $\alpha_f \rightarrow \alpha_{l'} \rightarrow \tau_f$ to x .²⁴ However, we do not need this flexibility. Therefore, we use $(\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_{l'} \rightarrow \tau_f$ as the type for x_f .

In addition to x_f , in order to support computation on x , we have interface cv_x .

The encoding for policy \mathcal{P} is described in Fig. 28. The additional (type and term) variables and conditions are highlighted in blue.

$$\begin{aligned} \Delta_{\mathcal{P}} &= \{\alpha_l \mid l \in \mathcal{L}\} \cup \{\alpha_l^f, \alpha^f \mid \exists x \in \mathbf{I}. \mathbf{F}(x) = \langle f, _ \rangle \wedge \text{lvl}(x) = l\} \\ \Gamma_{\mathcal{P}} &= \{x : \alpha_l \rightarrow \mathbf{int} \mid x \in \mathbf{I} \wedge \text{lvl}(x) = l \wedge x \notin \text{dom}(\mathbf{F})\} \cup \\ &\quad \{\text{cp}_l : \forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2 \mid l \in \mathcal{L}\} \cup \\ &\quad \{\text{cvu}_l^{l'} : \forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta) \mid l, l' \in \mathcal{L} \wedge l \sqsubseteq l'\} \cup \\ &\quad \{\text{wr}_l : \forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta \mid l \in \mathcal{L}\} \cup \\ &\quad \{x : \alpha_l^f \rightarrow \alpha^f, \quad x_f : (\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_{l'} \rightarrow \tau_f, \\ &\quad \text{cv}_x : (\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_l \rightarrow \mathbf{int} \mid x \in \mathbf{I} \wedge \mathbf{F}(x) = \langle f, l' \rangle \wedge \text{lvl}(x) = l\} \end{aligned}$$

Fig. 28. Contexts for declassification policies

Example 27 (Input declassified correctly). We consider the \mathcal{P}_{MOE} in Example 26. By using the contexts for \mathcal{P}_{MOE} , we can have the program $mi_f \ mi$ where the input mi at M is declassified correctly to L via the interface mi_f .

Example 28 (Computation on input that can be declassified). Since the type of mi is α_M^f , in order to do a computation on it, we must convert it as illustrated by the following program, where wrap_plus_one is of the type $\mathbf{int} \rightarrow \alpha_M \rightarrow \mathbf{int}$ (the description of this function is in Example 11).

$$\text{cp}_M[\mathbf{int}][\mathbf{int}] (\text{cv}_{mi} \ mi) \ \text{wrap_plus_one}$$

This program uses cv_{mi} to convert mi to a wrapped value and then transfer it to the continuation wrap_plus_one . In the context of the policy, this program is of the type $\alpha_M \rightarrow \mathbf{int}$, that is the output of the program is at M .

²⁴ Such functions can only handle x via x_f or handle x parametrically.

Indistinguishability As in §I.2, we first define environments w.r.t. an observer. The relational interpretation for α_l is as in the one for NI: α_l is interpreted as pairs of keys to open pairs of protected data. For α_l^f which corresponds to x that can be declassified to l' , observers that can observe values at l or l' can have the key.²⁵ Different from the interpretation for α_l , the interpretation for α_l^f directly captures the notion of indistinguishability for values declassified via f . In other words, the interpretations of α^f are different for different observers even though they have keys.

- if ζ can observe data at l (i.e. $l \sqsubseteq \zeta$), then indistinguishable values are equal values,
- if ζ can observe declassified data at l' but not at l (i.e. $l' \sqsubseteq \zeta$ and $l \not\sqsubseteq \zeta$), two values v_1 and v_2 are indistinguishable if they are indistinguishable via f ,
- otherwise, any two values are indistinguishable.

Let $R_f = \{\langle v_1, v_2 \rangle \mid \langle f\ v_1, f\ v_2 \rangle \in \llbracket \tau_f \rrbracket_\emptyset\}$, where \emptyset is the empty environment, be the relation s.t. two values are related if via f they behave the same. Let $id_{\mathbf{int}} = \{\langle n, n \rangle \mid n : \mathbf{int}\}$ be the identity relation on \mathbf{int} , and $Full_{\mathbf{int}} = \{\langle n, n' \rangle \mid \vdash n, n' : \mathbf{int}\}$ be the full relation on \mathbf{int} . The definition of environments w.r.t. an observer is as below.

Definition 14. *An environment ρ is an environment for \mathcal{P} w.r.t. an observer ζ (denoted by $\rho \models_\zeta \mathcal{P}$) if $\rho \models \Delta_{\mathcal{P}}$ and*

- for any $\alpha_l \in \Delta_{\mathcal{P}}$:

$$\rho(\alpha_l) = \begin{cases} \langle \mathbf{unit}, \mathbf{unit}, Full_{\mathbf{unit}} \rangle & \text{if } l \sqsubseteq \zeta, \\ \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle & \text{if } l \not\sqsubseteq \zeta, \end{cases}$$

- for any $\alpha_l^f \in \Delta_{\mathcal{P}}$ and $\alpha_{l'}^f \in \Delta_{\mathcal{P}}$ which corresponds to x s.t. $\mathbf{F}(x) = (f, l')$:

$$\rho(\alpha_l^f) = \begin{cases} \langle \mathbf{unit}, \mathbf{unit}, Full_{\mathbf{unit}} \rangle & \text{if } l \sqsubseteq \zeta \text{ or } l' \sqsubseteq \zeta \\ \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle & \text{otherwise,} \end{cases}$$

$$\rho(\alpha_{l'}^f) = \begin{cases} \langle \mathbf{int}, \mathbf{int}, id_{\mathbf{int}} \rangle & \text{if } l' \sqsubseteq \zeta, \\ \langle \mathbf{int}, \mathbf{int}, R_f \rangle & \text{if } l' \sqsubseteq \zeta \text{ and } l \not\sqsubseteq \zeta, \\ \langle \mathbf{int}, \mathbf{int}, Full_{\mathbf{int}} \rangle & \text{if } l' \not\sqsubseteq \zeta \text{ and } l \not\sqsubseteq \zeta. \end{cases}$$

We next define indistinguishability for \mathcal{P} as an instantiation of the logical relation. The definition is based on the environment of \mathcal{P} w.r.t. an observer ζ .

Definition 15. *Given a type τ s.t. $\Delta_{\mathcal{P}} \vdash \tau$. The indistinguishability relations on values and terms of τ for an observer ζ for \mathcal{P} (denoted by resp. $I_{\mathcal{P}}^\zeta \llbracket \tau \rrbracket$ and $I_{\mathcal{P}}^\zeta \llbracket \tau \rrbracket^{ev}$) are defined as*

$$I_{\mathcal{P}}^\zeta \llbracket \tau \rrbracket = \llbracket \tau \rrbracket_\rho \quad I_{\mathcal{P}}^\zeta \llbracket \tau \rrbracket^{ev} = \llbracket \tau \rrbracket_\rho^{ev}$$

where ρ is the environment for \mathcal{P} w.r.t. the observer ζ .

²⁵ If only observers who can observe values at l have the key, then to an observer at l' , all values at l are indistinguishable (see a similar explanation in Example 24).

Example 29 (Indistinguishability for \mathcal{P}_{MOE}). For \mathcal{P}_{MOE} presented in Example 26, the type of the input mi that can be declassified to L via f is $\alpha_M^f \rightarrow \alpha^f$. An observer H can observe values at M and hence, two values of mi are indistinguishable when they are equal. This is indeed captured in Def. 15 and Def. 14: the observer at H has the key to open wrapped values at M ($I_{\mathcal{P}_{MOE}}^H[\alpha_M^f] = \text{Full}_{\text{unit}}$), and two equal integer values are indistinguishable ($I_{\mathcal{P}_{MOE}}^H[\alpha^f] = id_{\text{int}}$).

An observer L can only observe values of mi after f . Therefore, the observer at L has the key to open values ($I_{\mathcal{P}_{MOE}}^L[\alpha_M^f] = \text{Full}_{\text{unit}}$) and two wrapped values n_1 and n_2 of mi are indistinguishable if $f(n_1) = f(n_2)$ as expressed by $I_{\mathcal{P}_{MOE}}^L[\alpha_M^f] = R_f = \{\langle n_1, n_2 \rangle \mid f(n_1) = f(n_2)\}$.

Typing implies security The implementations for x_f and cv_x are as below:

$$\begin{aligned} \text{dec}_f &= \lambda x : \text{unit} \rightarrow \text{int}. \lambda _ : \text{unit}. f(x \langle \rangle) \\ \text{conv} &= \lambda x : \text{unit} \rightarrow \text{int}. x \end{aligned}$$

Definition 16. An environment ρ is a full environment for \mathcal{P} w.r.t. an observer ζ (denoted by $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$) if $\rho|_{tv} \models_{\zeta} \mathcal{P}$ and

- for all x of the type $\alpha_l \rightarrow \text{int}$, $\rho(x) \in I_{\mathcal{P}}^{\zeta}[\alpha_l \rightarrow \text{int}]$,
- for all l , $\rho(\text{cp}_l) = \langle \text{comp}, \text{comp} \rangle$,
- for all l and l' s.t. $l \sqsubset l'$, $\rho(\text{cvu}_l') = \langle \text{convup}, \text{convup} \rangle$,
- for all x of the type $\alpha_l^f \rightarrow \alpha^f$, $\rho(x) \in I_{\mathcal{P}}^{\zeta}[\alpha_l^f \rightarrow \alpha^f]$,
- for all x_f , $\rho(x_f) = \langle \text{dec}_f, \text{dec}_f \rangle$,
- for all cv_x , $\rho(cv_x) = \langle \text{conv}, \text{conv} \rangle$.

In order to instantiate the abstraction theorem to prove security, we need to prove that comp , convup , dec_f , conv are indistinguishable to themselves for any observer.

Lemma 40. For any ζ , it follows that:

$$\begin{aligned} \langle \text{comp}, \text{comp} \rangle &\in I_{\mathcal{P}}^{\zeta}[\forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2] \\ \langle \text{convup}, \text{convup} \rangle &\in I_{\mathcal{P}}^{\zeta}[\forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta)] \\ \langle \text{wrap}, \text{wrap} \rangle &\in I_{\mathcal{P}}^{\zeta}[\forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta] \\ \langle \text{conv}, \text{conv} \rangle &\in I_{\mathcal{P}}^{\zeta}[(\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_l \rightarrow \text{int}] \end{aligned}$$

Lemma 41. For any ζ , for any α_l^f and α^f corresponding to x s.t. $lv_l(x) = l$ and $\mathbf{F}(x) = \langle f, l' \rangle$:

$$\langle \text{dec}_f, \text{dec}_f \rangle \in I_{\mathcal{P}}^{\zeta}[(\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_{l'} \rightarrow \tau_f].$$

We next prove that if a program is well-typed in $\Delta_{\mathcal{P}}$, $\Gamma_{\mathcal{P}}$, then it is type-based relax noninterferent w.r.t. \mathcal{P} , that is it transforms indistinguishable inputs to indistinguishable outputs.

Theorem 11. *If $\Delta_{\mathcal{P}}, \Gamma_{\mathcal{P}} \vdash e : \tau$, then for any $\zeta \in \mathcal{L}$ and $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$,*

$$\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_{\mathcal{P}}^{\zeta}[\tau]^{ev}.$$

Proof. Since $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$, from the definition of $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$, Lemma 40, and Lemma 41, it follows that $\rho \models_{\zeta} \Delta_{\mathcal{P}}, \Gamma_{\mathcal{P}}$. Since $\Delta_{\mathcal{P}}, \Gamma_{\mathcal{P}} \vdash e : \tau$, from Theorem 8, we have that $\langle \rho_L(e), \rho_R(e) \rangle \in \llbracket \tau \rrbracket_{\rho}^{ev}$. From the definition of indistinguishability, it follows that $\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_{\mathcal{P}}^{\zeta}[\tau]^{ev}$.

Corollary 3 (RNI for free). *If $\Delta_{\mathcal{P}}, \Gamma_{\mathcal{P}} \vdash e : (\alpha_{l_1} \rightarrow \mathbf{int}) \times \cdots \times (\alpha_{l_n} \rightarrow \mathbf{int})$, then for any $\zeta \in \mathcal{L}$ and $\rho \models_{\zeta}^{\text{full}} \mathcal{P}$, it is that $\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_{\mathcal{P}}^{\zeta}[(\alpha_{l_1} \rightarrow \mathbf{int}) \times \cdots \times (\alpha_{l_n} \rightarrow \mathbf{int})]^{ev}$.*

I.4 Extensions

Similar to the encoding in §4, the encoding in this section can also be extended to support richer policies. The ideas behind the extensions are similar to the ones in §A. Here we only present two extensions: multiple declassification functions for an input and more inputs involved in a declassification functions (global policies). The remaining extension is about declassifying via equivalent functions can be obtained by combining the idea in §A and §I.4.

More declassification functions In general an input can be declassified in more than one way. To show how this can be accommodated, we present an extension for a policy \mathcal{P}_{M1} defined on the lattice $\langle \mathcal{L}_{\diamond}, \sqsubseteq \rangle$, where $\mathcal{L}_{\diamond} = \{H, M_1, M_2, L\}$ and $L \sqsubseteq M_i \sqsubseteq H$ (for $i \in \{1, 2\}$). The input set is $\mathbf{I} = \{hi, mi1\}$, where inputs hi and $mi1$ are at resp. H and M_1 .

The policy allows an input to be declassified via multiple functions to different levels. Specifically, the policy allows that:

- input hi can be declassified via f_1 to M_1 or f_2 to M_2 for some f_1 and f_2 , where $\vdash f_1 : \mathbf{int} \rightarrow \tau_{f_1}$ and $\vdash f_2 : \mathbf{int} \rightarrow \tau_{f_2}$,
- input $mi1$ can be declassified via g_1 or g_2 to L for some g_1 and g_2 , where $\vdash g_1 : \mathbf{int} \rightarrow \tau_{g_1}$ and $\vdash g_2 : \mathbf{int} \rightarrow \tau_{g_2}$

The policy however, does not allow hi to be declassified to L via g and f_i .

For this policy, we have the context described in Fig. 29 where the details of interfaces \mathbf{cp}_l , \mathbf{cvu}_l' and \mathbf{wr}_l are as in Fig. 28 and are omitted. As in §I.3, we have new type variables for inputs hi and $mi1$ and interfaces hi_{f_1} , hi_{f_2} , $mi1_{g_1}$, $mi1_{g_2}$ for declassification functions.

We next define the full environment ρ for \mathcal{P}_{M1} w.r.t. an observer ζ (denoted by $\rho \models_{\zeta}^{\text{full}} \mathcal{P}_{M1}$). For such a ρ , the mapping for term variables is as in Def. 16. For a type variable $\alpha_l \in \Delta_{\mathcal{P}_{M1}}$, $\rho(\alpha_l)$ is as in Def. 14. For $\alpha_H^{f_1, f_2}$, an observer at M_1 , M_2 or H has the key.

$$\rho(\alpha_H^{f_1, f_2}) = \begin{cases} \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle & \text{if } obs \in \{M_1, M_2, H\} \\ \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\Delta_{\mathcal{P}_{M1}} &= \{\alpha_l \mid l \in \mathcal{L}_\diamond\} \cup \{\alpha_H^{f_1, f_2}, \alpha^{f_1, f_2}, \alpha_{M_1}^{g_1, g_2}, \alpha^{g_1, g_2}\} \\
\Gamma_{\mathcal{P}_{M1}} &= \{\mathbf{cp}_l : \dots\} \cup \{\mathbf{cvu}_l' : \dots\} \cup \{\mathbf{wr}_l : \dots\} \cup \\
&\quad \{hi : \alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}\} \cup \\
&\quad \{hi_{f_1} : (\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_{M_1} \rightarrow \tau_{f_1}, \quad hi_{f_2} : (\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_{M_2} \rightarrow \tau_{f_2}\} \cup \\
&\quad \{\mathbf{cv}_{hi} : (\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_H \rightarrow \mathbf{int}\} \cup \\
&\quad \{mi1 : \alpha_{M_1}^{g_1, g_2} \rightarrow \alpha^{g_1, g_2}\} \cup \\
&\quad \{mi1_{g_1} : (\alpha_{M_1}^{g_1, g_2} \rightarrow \alpha^{g_1, g_2}) \rightarrow \alpha_L \rightarrow \tau_{g_1}, \quad mi1_{g_2} : (\alpha_{M_1}^{g_1, g_2} \rightarrow \alpha^{g_1, g_2}) \rightarrow \alpha_L \rightarrow \tau_{g_2}\} \cup \\
&\quad \{\mathbf{cv}_{mi1} : (\alpha_{M_1}^{g_1, g_2} \rightarrow \alpha^{g_1, g_2}) \rightarrow \alpha_{M_1} \rightarrow \mathbf{int}\}
\end{aligned}$$

Fig. 29. Contexts for policy \mathcal{P}_{M1}

For α_{f_1, f_2} , the definition is straightforward.

$$\rho(\alpha_{f_1, f_2}) = \begin{cases} \langle \mathbf{int}, \mathbf{int}, id_{\mathbf{int}} \rangle & \text{if } \zeta = H, \\ \langle \mathbf{int}, \mathbf{int}, R_{f_i} \rangle & \text{if } \zeta = M_i, \\ \langle \mathbf{int}, \mathbf{int}, \mathbf{Full}_{\mathbf{int}} \rangle & \text{if } \zeta = L. \end{cases}$$

For $\alpha_{M_1 g_1, g_2}$, all observers have the key (since the input can be declassified to L).

$$\rho(\alpha_{M_1}^{g_1, g_2}) = \langle \mathbf{unit}, \mathbf{unit}, \mathbf{Full}_{\mathbf{unit}} \rangle$$

For α^{g_1, g_2} , when ζ is L , since this observer can apply g_1 and g_2 to $mi1$, two wrapped values are indistinguishable if they cannot be distinguished by both g_1 and g_2 . In addition, since data at L can flow to M_2 , for an observer at M_2 , two wrapped values at M_1 are indistinguishable if they are indistinguishable at L . Therefore, $\rho(\alpha^{g_1, g_2})$ is as below:

$$\rho(\alpha^{g_1, g_2}) = \begin{cases} \langle \mathbf{int}, \mathbf{int}, id_{\mathbf{int}} \rangle & \text{if } M_1 \sqsubseteq \zeta, \\ \langle \mathbf{int}, \mathbf{int}, R_{g_1, g_2} \rangle & \text{if } M_1 \not\sqsubseteq \zeta, \end{cases}$$

where $R_{g_1, g_2} = \{(n_1, n_2) \mid (g_1 \ n_1, g_1 \ n_2) \in \llbracket \tau_{g_1} \rrbracket_\emptyset^{\mathbf{ev}} \wedge (g_2 \ n_1, g_2 \ n_2) \in \llbracket \tau_{g_2} \rrbracket_\emptyset^{\mathbf{ev}}\}$.

We define indistinguishability w.r.t. an ζ as an instantiation of the logical relation with an arbitrary $\rho \models_\zeta^{\mathbf{full}} \mathcal{P}_{M1}$. The implementations of \mathbf{cp}_- , \mathbf{cv}_- , \mathbf{cvu}_l' , hi_{f_i} , and $mi1_{g_i}$ are as in §I.3. We also have that these implementations are indistinguishable to themselves for any observer. Therefore, from the abstraction theorem, we again obtain that for any program e , if $\Delta_{\mathcal{P}_{M1}}, \Gamma_{\mathcal{P}_{M1}} \vdash e : \tau$, then this program maps indistinguishable inputs to indistinguishable outputs. Proofs are in §J.3.

For example, we consider programs $e_{E1} = hi_{f_1} \ hi$ and $e_{E2} = \langle mi1_{g_1} \ mi1 \rangle$. These programs are well-typed in the context of \mathcal{P}_{M1} , and their types are respectively $\alpha_M \rightarrow \tau_{f_1}$ and $\alpha_L \rightarrow \tau_{g_1}$ and hence, on indistinguishable inputs, their outputs are indistinguishable at resp. M_1 and L .

Global policies We now consider policies where a declassifier can involve more than one input. For simplicity, in this subsection, we consider a policy \mathcal{P}_{Av} defined on the lattice $\langle \mathcal{L}_\diamond, \sqsubseteq \rangle$ described in §I.4. There are two inputs: $mi1$ and $mi2$ at respectively M_1 and M_2 . The average of these inputs can be declassified to L , i.e. they can be declassified to L via $f = \lambda x : \mathbf{int}_1 \times \mathbf{int}_2. (\pi_1 x + \pi_2 x)/2$.²⁶ Notice that here we use subscripts for the input type of f to mean that the confidential inputs $mi1$ and $mi2$ are corresponding to resp. the first and second elements of an input of f .

To encode the requirement that $mi1$ and $mi2$ can be declassified via f , we introduce a new variable y , which is corresponding to the tuple of inputs $mi1$ and $mi2$.²⁷ Individual inputs cannot be declassified, and only y can be declassified via f . Thus, we have the context described in Fig. 30, where the details of interfaces \mathbf{cp}_l , $\mathbf{cvu}_l^{f'}$ and \mathbf{wr}_l are as in Fig. 28 and are omitted. Note that we introduce new type variables α^f and α_{M_1, M_2}^f , where α_{M_1, M_2}^f is the type of the key to open y . Since $mi1$ cannot be declassified directly, its type is $\alpha_{M_1} \rightarrow \mathbf{int}$. Similarly, the type of $mi2$ is $\alpha_{M_2} \rightarrow \mathbf{int}$.

$$\begin{aligned} \Delta_{\mathcal{P}_{Av}} &= \{\alpha_l \mid l \in \mathcal{L}_\diamond\} \cup \{\alpha^f, \alpha_{M_1, M_2}^f\} \\ \Gamma_{\mathcal{P}_{Av}} &= \{\mathbf{cp}_l : \dots\} \cup \{\mathbf{cvu}_l^{f'} : \dots\} \cup \{\mathbf{wr}_l : \dots\} \cup \\ &\quad \{y : \alpha_{M_1, M_2}^f \rightarrow \alpha^f, \quad y_f : (\alpha_{M_1, M_2}^f \rightarrow \alpha^f) \rightarrow \alpha_L \rightarrow \mathbf{int}\} \cup \\ &\quad \{mi1 : \alpha_{M_1} \rightarrow \mathbf{int}, \quad mi2 : \alpha_{M_2} \rightarrow \mathbf{int}\} \end{aligned}$$

Fig. 30. Contexts for policy \mathcal{P}_{Av}

Environment for \mathcal{P}_{Av} . We next define an environment for \mathcal{P}_{Av} (denoted by $\rho \models_\zeta \mathcal{P}_{Av}$). The definition for $\rho \models_\zeta \mathcal{P}_{Av}$ is similar to Def. 14, except for α_{M_1, M_2}^f and α^f .

For α_{M_1, M_2}^f , since y can be declassified to L , all observers have the key.

$$\rho(\alpha_{M_1, M_2}^f) = \langle \mathbf{unit}, \mathbf{unit}, \mathbf{Full}_{\mathbf{unit}} \rangle$$

Since α^f is the type of y which is corresponding to both inputs, its concrete type is $\mathbf{int} \times \mathbf{int}$.

- When $\zeta = H$ (i.e. the observer ζ can observe both inputs at M_1 and M_2), the interpretation of α^f is just $id_{\mathbf{int} \times \mathbf{int}}$.

²⁶ We can extend the encoding presented in this section to have policies where different subsets of \mathbf{I} can be declassified and to have more than one declassifier associated with a set of confidential inputs.

²⁷ This idea can be generalized to capture the requirement in which there are more than two inputs that can be declassified. When there are n inputs that can be declassified, we just introduce a fresh variable y which is correspond to the n -tuple of inputs.

- When $\zeta = L$, since the observer can apply f on y , two tuples of inputs are indistinguishable if the results of f on them are the same.
- When $\zeta = M_i$, since declassified data at L can be observed at M_i , two tuples of inputs are indistinguishable if they are indistinguishable at L .

Therefore, we define $\rho(\alpha_f)$ as below:

$$\rho(\alpha_f) = \begin{cases} \langle \mathbf{int} \times \mathbf{int}, \mathbf{int} \times \mathbf{int}, id_{\mathbf{int} \times \mathbf{int}} \rangle & \text{if } \zeta = H, \\ \langle \mathbf{int} \times \mathbf{int}, \mathbf{int} \times \mathbf{int}, R_f^\bullet \rangle & \text{otherwise,} \end{cases}$$

where $R_f^\bullet = \{ \langle \langle v, u \rangle, \langle v', u' \rangle \rangle \mid \langle f \langle v, u \rangle, f \langle v', u' \rangle \rangle \in \llbracket \mathbf{int} \rrbracket_\emptyset^{\text{ev}} \}$.

Full environment for \mathcal{P}_{Av} . Different from previous sections, in order to define full environments, we need to encode the correspondence between inputs and argument of the declassifier. We say that an environment ρ of \mathcal{P}_{Av} is *consistent* w.r.t. an ζ if $\rho \models_\zeta \mathcal{P}_{Av}$ and

$$\begin{aligned} \pi_1(\rho_L(y) \langle \rangle) &= \rho_L(mi1) \langle \rangle & \pi_2(\rho_L(y) \langle \rangle) &= \rho_L(mi2) \langle \rangle \\ \pi_1(\rho_R(y) \langle \rangle) &= \rho_R(mi1) \langle \rangle & \pi_2(\rho_R(y) \langle \rangle) &= \rho_R(mi2) \langle \rangle \end{aligned}$$

This additional condition takes care of the correspondence of inputs and the arguments of the designated declassifier.

We construct dec_f which is used as the concrete input for y_f .

$$\lambda x : \mathbf{unit} \rightarrow \mathbf{int} \times \mathbf{int}. \lambda_- : \mathbf{unit}. f(x \langle \rangle)$$

An environment ρ is full for \mathcal{P}_{Av} w.r.t. an ζ (denoted by $\rho \models_\zeta^{\text{full}} \mathcal{P}_{Av}$) if ρ is consistent w.r.t. ζ , it maps y_f to $\langle \text{dec}_f, \text{dec}_f \rangle$, and the mapping of other term variables is as in Def. 16.

We then define indistinguishability as an instantiation of the logical relation as in Def. 15. We also have the free theorem saying that if $\Delta_{\mathcal{P}_{Av}}, \Gamma_{\mathcal{P}_{Av}} \vdash e : \tau$, then e maps indistinguishable inputs to indistinguishable outputs. The proof goes through without changes.

We illustrate this section by considering program y_f where the declassifier is applied correctly. This program is well-typed in the contexts for \mathcal{P}_{Av} and its type is $\alpha_L \rightarrow \mathbf{int}$ and hence, this program maps indistinguishable inputs to indistinguishable outputs.

Remark 4. We may use two type variables α_1 and α_2 for the two inputs (i.e. $mi_1 : \alpha_1$ and $mi_2 : \alpha_2$) and use $y_f : \alpha_1 \times \alpha_2 \rightarrow \mathbf{int}$ for the declassifier. Since we used two type variables separately, we may define the indistinguishability relations for them as the full relation on \mathbf{int} . However, when we define the indistinguishability relations separately, the declassifier f may not be related to itself at $\alpha_1 \times \alpha_2 \rightarrow \mathbf{int}$.²⁸ For example, we have 3 and 4 are indistinguishable at α_1 , 4 and 5 are indistinguishable at α_2 but $f \langle 3, 5 \rangle \neq f \langle 4, 6 \rangle$.

²⁸ In order to use the abstraction theorem to prove security, we need that the declassifier f is indistinguishable to itself at $\alpha_1 \times \alpha_2 \rightarrow \mathbf{int}$.

To have f related to itself, the indistinguishability relations for α_1 and α_2 should depend on each other. For example, suppose that v_1 and v'_1 are indistinguishable values for mi_1 . Then two values v_2 and v'_2 for mi_2 are indistinguishable when $f\langle v_1, v_2 \rangle = f\langle v'_1, v_2 \rangle$. In other words, whether v_2 and v'_2 are indistinguishable depends on v_1 and v'_1 . However, we have difficulty to express such dependency between α_1 and α_2 in the language presented in §I.1 and hence, we introduce a new variable y .

J Proofs for multi-level encodings

J.1 Proofs for the encoding for NI

Lemma 37. Suppose that $\rho \models \Delta$ for some Δ . It follows that:

- if $\langle v_1, v_2 \rangle \in \llbracket \tau \rrbracket_\rho$, then $\vdash v_1 : \rho_L(\tau)$, $\vdash v_2 : \rho_R(\tau)$ and
- if $\langle e_1, e_2 \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}$, then $\vdash e_1 : \rho_L(\tau)$ and $\vdash e_2 : \rho_R(\tau)$.

Proof. The second part of the lemma follows directly from rule FR-Term. We prove the first part of the lemma by induction on structure of τ .

Case 1: int. We consider $\langle v_1, v_2 \rangle \in \llbracket \text{int} \rrbracket_\rho$. From FR-Int, we have that $\vdash v_i : \text{int}$. Since $\rho_L(\text{int}) = \rho_R(\text{int}) = \text{int}$, we have that $\vdash v_1 : \rho_L(\text{int})$ and $\vdash v_2 : \rho_R(\text{int})$.

Case 2: unit. The proof is similar to the one of Case 1. We consider $\langle v_1, v_2 \rangle \in \llbracket \text{int} \rrbracket_\rho$. From FR-Int, we have that $\vdash v_i : \text{int}$. Since $\rho_L(\text{int}) = \rho_R(\text{int}) = \text{int}$, we have that $\vdash v_1 : \rho_L(\text{int})$ and $\vdash v_2 : \rho_R(\text{int})$.

Case 3: α . We consider $\langle v_1, v_2 \rangle \in \llbracket \alpha \rrbracket_\rho$. From the FR-Var rule, $\langle v_1, v_2 \rangle \in \rho(\alpha) \in \text{Rel}(\tau_1, \tau_2)$. From the definition of $\text{Rel}(\tau_1, \tau_2)$, we have that $\vdash v_1 : \rho_L(\alpha)$ and $\vdash v_2 : \rho_R(\alpha)$.

Case 4: $\tau_1 \times \tau_2$. We consider $\langle v_1, v_2 \rangle \in \llbracket \tau_1 \times \tau_2 \rrbracket_\rho$. We then have that $v_1 = \langle v_{11}, v_{12} \rangle$ for some v_{11} and v_{12} and $v_2 = \langle v_{21}, v_{22} \rangle$ for some v_{21} and v_{22} . From FR-Pair, it follows that $\langle v_{11}, v_{21} \rangle \in \llbracket \tau_1 \rrbracket_\rho$ and $\langle v_{12}, v_{22} \rangle \in \llbracket \tau_2 \rrbracket_\rho$. From IH (on τ_1 and τ_2), we have that $\vdash v_{11} : \rho_L(\tau_1)$, $\vdash v_{21} : \rho_R(\tau_1)$, $\vdash v_{12} : \rho_L(\tau_2)$, and $\vdash v_{22} : \rho_R(\tau_2)$. From FT-Pair, $\vdash \langle v_{11}, v_{12} \rangle : \rho_L(\tau_1) \times \rho_L(\tau_2)$ and $\vdash \langle v_{21}, v_{22} \rangle : \rho_R(\tau_1) \times \rho_R(\tau_2)$. Thus, $\vdash v_1 : \rho_L(\tau_1) \times \rho_L(\tau_2)$ and $\vdash v_2 : \rho_R(\tau_1) \times \rho_R(\tau_2)$. In other words, $\vdash v_1 : \rho_L(\tau_1 \times \tau_2)$ and $\vdash v_2 : \rho_R(\tau_1 \times \tau_2)$.

Case 5: $\tau_1 \rightarrow \tau_2$. We consider $\langle v_1, v_2 \rangle \in \llbracket \tau_1 \rightarrow \tau_2 \rrbracket_\rho$. We now look at arbitrary $\langle v'_1, v'_2 \rangle \in \llbracket \tau_1 \rrbracket_\rho$. From FR-Fun, it follows that $\langle v_1 v'_1, v_2 v'_2 \rangle \in \llbracket \tau_2 \rrbracket_\rho^{\text{ev}}$. From IH on τ_1 and the second part of the lemma on τ_2 , we have that $\vdash v'_1 : \rho_L(\tau_1)$, $\vdash v'_2 : \rho_R(\tau_1)$, $\vdash v_1 v'_1 : \rho_L(\tau_2)$, and $\vdash v_2 v'_2 : \rho_R(\tau_2)$. From FT-App, we have that $\vdash v_1 : \rho_L(\tau_1 \rightarrow \tau_2)$ and $\vdash v_2 : \rho_R(\tau_1 \rightarrow \tau_2)$.

Case 6: $\forall \alpha. \tau$. We consider an arbitrary $R \in \text{Rel}(\tau_1, \tau_2)$. Since $\langle v_1, v_2 \rangle \in \llbracket \forall \alpha. \tau \rrbracket_\rho$, we have that $\langle v_1[\tau_1], v_2[\tau_2] \rangle \in \llbracket \tau \rrbracket_{\rho[(\tau_1, \tau_2, R)/\alpha]}^{\text{ev}}$. From IH, $\vdash v_1[\tau_1] : \rho'_L(\tau)$, where $\rho' = \rho[(\tau_1, \tau_2, R)/\alpha]$. From the typing rule, $\vdash v_1 : \rho'_L(\forall \alpha. \tau)$. Since α is bound in τ , we have that $\vdash v_1 : \rho_L(\forall \alpha. \tau)$. Similarly, $\vdash v_2 : \rho_R(\forall \alpha. \tau)$.

Lemma 38 - Part 1. For any ζ , it follows that:

$$\langle \text{comp}, \text{comp} \rangle \in \hat{F}_{NI}[\forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2].$$

Proof. Let $\rho \models_{\zeta} \text{NI}$. We first prove for cp_l :

$$\rho(\text{cp}_l) \in I_{NI}^{\zeta} \llbracket \forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2 \rrbracket.$$

From the definition of indistinguishability, we need to prove that:

$$\rho(\text{cp}_l) \in \llbracket \forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2 \rrbracket_{\rho},$$

where $\rho \models_{\zeta} \text{NI}$. From the definition of the logical relation, we need to prove that for any $R \in \text{Rel}(\tau_1, \tau'_1)$ and $S \in \text{Rel}(\tau_2, \tau'_2)$, we have that:

$$\langle \text{comp}[\tau_1][\tau_2], \text{comp}[\tau'_1][\tau'_2] \rangle \in \llbracket (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2 \rrbracket_{\rho'}^{\text{ev}}$$

where $\rho' = \rho, \beta_1 \mapsto \langle \tau_1, \tau'_1, R \rangle, \beta_2 \mapsto \langle \tau_2, \tau'_2, S \rangle$.

We now need to prove that for all $\langle v, v' \rangle \in \llbracket (\alpha_l \rightarrow \beta_1) \rrbracket_{\rho'}$, for all $\langle f, f' \rangle \in \llbracket \beta_1 \rightarrow (\alpha_l \rightarrow \beta_2) \rrbracket_{\rho'}$, it follows that

$$\langle f(v \langle \rangle), f'(v' \langle \rangle) \rangle \in \llbracket \alpha_l \rightarrow \beta_2 \rrbracket_{\rho'}^{\text{ev}}.$$

We now have two cases:

- $l \not\sqsubseteq \zeta$. We have that $\rho(\alpha_l) = \langle \text{unit}, \text{unit}, \emptyset \rangle$ and hence, $\langle f(v \langle \rangle), f'(v' \langle \rangle) \rangle \in \llbracket \alpha_l \rightarrow \beta_2 \rrbracket_{\rho'}^{\text{ev}}$ holds vacuously.
- $l \sqsubseteq \zeta$. We have that $\rho(\alpha_l) = \langle \text{unit}, \text{unit}, \text{Full}_{\text{unit}} \rangle$. Since $\langle f, f' \rangle \in \llbracket \beta_1 \rightarrow (\alpha_l \rightarrow \beta_2) \rrbracket_{\rho'}$, we just need to prove that $\langle v \langle \rangle, v' \langle \rangle \rangle \in \llbracket \beta_1 \rrbracket_{\rho'}^{\text{ev}}$. And this follows from the fact that $\langle v, v' \rangle \in \llbracket \alpha_l \rightarrow \beta_1 \rrbracket_{\rho'}$.

Lemma 38 - Part 2. For any ζ , it follows that:

$$\langle \text{convup}, \text{convup} \rangle \in I_{NI}^{\zeta} \llbracket \forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta) \rrbracket.$$

Proof. We need to prove that for any $\rho \models_{\zeta} \text{NI}$,

$$\langle \text{convup}, \text{convup} \rangle \in \llbracket \forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta) \rrbracket_{\rho}.$$

That is for any closed types τ and τ' , for any $R \in \text{Rel}(\tau, \tau')$,

$$\langle \text{convup}[\sigma], \text{convup}[\sigma'] \rangle \in \llbracket (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta) \rrbracket_{\rho'},$$

where $\rho' = \rho, \beta \mapsto \langle \tau, \tau', R \rangle$.

From the definition of **convup**, we need to prove that for any $(v, v') \in \llbracket \alpha_l \rightarrow \beta \rrbracket_{\rho'}$, it follows that $(v, v') \in \llbracket \alpha_{l'} \rightarrow \beta \rrbracket_{\rho'}$. We have the following cases:

- $l' \sqsubseteq \zeta$. Since $l \sqsubset l'$, we have that $l \sqsubset l' \sqsubseteq \zeta$. Since $\rho \models_{\zeta} \text{NI}$, we have that $\rho(\alpha_l) = \rho(\alpha'_l) = \langle \text{unit}, \text{unit}, \text{Full}_{\text{unit}} \rangle$. Thus, for any $\langle v, v' \rangle \in \llbracket \alpha_l \rightarrow \beta \rrbracket_{\rho'}$, We have that $\langle v, v' \rangle \in \llbracket \alpha_{l'} \rightarrow \beta \rrbracket_{\rho'}$.
- $l' \not\sqsubseteq \zeta$. We have that $\rho(\alpha'_l) = \langle \text{unit}, \text{unit}, \emptyset \rangle$. Since the interpretation of $\alpha_{l'}$ is \emptyset , for any $\langle v, v' \rangle \in \llbracket \alpha_l \rightarrow \beta \rrbracket_{\rho'}$, we have that $\langle v, v' \rangle \in \llbracket \alpha_{l'} \rightarrow \beta \rrbracket_{\rho'}$.

Lemma 38 - Part 3. For any ζ , it follows that:

$$\langle \text{wrap}, \text{wrap} \rangle \in I_{NI}^\zeta \llbracket \forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta \rrbracket.$$

Proof. We need to prove that for any $\rho \models_\zeta \text{NI}$,

$$\langle \text{wrap}, \text{wrap} \rangle \in \llbracket \forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta \rrbracket_\rho$$

That is for any τ_1, τ_2 and R s.t. $R \in \text{Rel}(\tau_1, \tau_2)$,

$$\langle \text{wrap}[\tau_1], \text{wrap}[\tau_2] \rangle \in \llbracket \beta \rightarrow \alpha_l \rightarrow \beta \rrbracket_{\rho, \beta \mapsto \langle \tau_1, \tau_2, R \rangle}$$

Let $\rho' = \rho, \beta \mapsto \langle \tau_1, \tau_2, R \rangle$, we need to prove that for any $(v_1, v_2) \in R$, we have that:

$$\langle \text{wrap}[\tau_1] v_1, \text{wrap}[\tau_2] v_2 \rangle \in \llbracket \alpha_l \rightarrow \beta \rrbracket_{\rho'}.$$

From the definition of **wrap**, we need to prove that:

$$\langle \lambda_- : \mathbf{unit}.v_1, \lambda_- : \mathbf{unit}.v_2 \rangle \in \llbracket \alpha_l \rightarrow \beta \rrbracket_{\rho'}.$$

When $l \not\sqsubseteq \zeta$, $\rho'(\alpha_l) = \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle$ and hence, the statement hold vacuously. When $l \sqsubseteq \zeta$, $\rho'(\alpha_l) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$ and hence, we only need to prove that $\langle v_1, v_2 \rangle \in R$. This is trivial from the assumption that $\langle v_1, v_2 \rangle \in R$.

Lemma 39. For any closed types τ and τ' :

1. for any e s.t. $\vdash e : \tau$, for any $f : \tau \rightarrow \mathbf{unit} \rightarrow \tau'$,

$$\text{comp}[\tau][\tau'] (\text{wrap}[\tau] e) f \cong f e$$

2. for any e s.t. $\vdash e : \mathbf{unit} \rightarrow \tau$,

$$\text{comp}[\tau][\tau'] e \text{wrap}[\tau] \cong e$$

3. for any e s.t. $\vdash e : \mathbf{unit} \rightarrow \tau$, $\vdash f : \tau \rightarrow \mathbf{unit} \rightarrow \tau''$, $\vdash g : \tau'' \rightarrow \mathbf{unit} \rightarrow \tau$

$$\text{comp}[\tau''][\tau'] (\text{comp}[\tau][\tau''] e f) g \cong \text{comp}[\tau][\tau'] e (\lambda x : \tau. \text{comp}[\tau''][\tau'] (f x) g)$$

Proof. We prove (1). From the definition of **wrap**, $\text{wrap}[\tau] e \rightarrow^* \lambda_- : \mathbf{unit}.e$. From the definition of **comp**, we have that:

$$\text{comp}[\tau][\tau'] (\text{wrap}[\tau] e) f \rightarrow^* f(\lambda_- : \mathbf{unit}.e \langle \rangle) \rightarrow^* f(e)$$

Thus, $\text{comp}[\tau][\tau'] (\text{wrap}[\tau] e) f \cong f e$.

We prove (2). From the implementation of **wrap**, $\text{wrap}[\tau] \rightarrow^* \lambda x : \tau. \lambda_- : \mathbf{unit}.x$. Let v be the value s.t. $e \rightarrow^* v$ (note that in our language, all closed terms can be reduced to values). Let v' of the type τ s.t. $(v \langle \rangle) \rightarrow^* v'$. Therefore, we have that $(e \langle \rangle) \rightarrow^* v \langle \rangle \rightarrow^* v'$.

From the implementation of **comp**,

$$\text{comp}[\tau][\tau'] e \text{wrap}[\tau] \rightarrow^* (\lambda x : \tau. \lambda_- : \mathbf{unit}.x)(v \langle \rangle) \rightarrow^* \lambda_- : \mathbf{unit}.v'$$

Thus, $(\text{comp}[\tau][\tau'] e \text{wrap}[\tau]) \langle \rangle \rightarrow^* v'$. As proven above, $(e \langle \rangle) \rightarrow^* v \langle \rangle \rightarrow^* v'$. Thus, $\text{comp}[\tau][\tau'] e \text{wrap}[\tau] \cong e$.

We prove (3). Let v, v''', v'' and v' be values s.t.

- v is of type $\mathbf{unit} \rightarrow \tau$ and $e \rightarrow^* v$,
- v''' is of type τ and $v \langle \rangle \rightarrow^* v'''$,
- v'' is of the type $\mathbf{unit} \rightarrow \tau''$ s.t. $f v''' \rightarrow^* v''$
- v' is of the type $\mathbf{unit} \rightarrow \tau'$ s.t. $g(v'' \langle \rangle) \rightarrow^* v'$.

We first look at $\text{comp}[\tau''][\tau'] (\text{comp}[\tau][\tau''] e f) g$. We have that

$$\text{comp}[\tau][\tau''] e f \rightarrow^* f(v \langle \rangle) \rightarrow^* f(v''') \rightarrow^* v''.$$

Thus, $\text{comp}[\tau''][\tau'] (\text{comp}[\tau][\tau''] e f) g \rightarrow^* g(v'' \langle \rangle) \rightarrow^* v'$.

We now look at $\text{comp}[\tau][\tau'] e (\lambda x : \tau. \text{comp}[\tau''][\tau'] (f x) g)$. We need to prove that $\text{comp}[\tau][\tau'] e (\lambda x : \tau. \text{comp}[\tau''][\tau'] (f x) g) \rightarrow^* v'$. We have that:

$$\begin{aligned} \text{comp}[\tau][\tau'] e (\lambda x : \tau. \text{comp}[\tau''][\tau'] (f x) g) &\rightarrow^* (\lambda x : \tau. \text{comp}[\tau''][\tau'] (f x) g)(v \langle \rangle) \\ &\rightarrow^* (\lambda x : \tau. \text{comp}[\tau''][\tau'] (f x) g)(v''') \\ &\rightarrow^* \text{comp}[\tau''][\tau'] (f v''') g \\ &\rightarrow^* g(v'' \langle \rangle) \rightarrow^* v' \end{aligned}$$

J.2 Proofs for the encoding for TRNI

Lemma 40. For any ζ , it follows that:

$$\begin{aligned} \langle \text{comp}, \text{comp} \rangle &\in I_{\mathcal{P}}^{\zeta} [\forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2] \\ \langle \text{convup}, \text{convup} \rangle &\in I_{\mathcal{P}}^{\zeta} [\forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta)] \\ \langle \text{wrap}, \text{wrap} \rangle &\in I_{\mathcal{P}}^{\zeta} [\forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta] \\ \langle \text{conv}, \text{conv} \rangle &\in I_{\mathcal{P}}^{\zeta} [(\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_l \rightarrow \mathbf{int}] \end{aligned}$$

Proof. The proofs for the first three parts are similar to the corresponding ones in Lemma 38. We now prove the last part. Let ρ be an environment s.t. $\rho \models_{\zeta} \mathcal{P}$. From Def. 15, we need to prove that $\langle \text{conv}, \text{conv} \rangle \in \llbracket (\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_l \rightarrow \mathbf{int} \rrbracket_{\rho}$. That is for any $(v, v') \in \llbracket \alpha_l^f \rightarrow \alpha^f \rrbracket_{\rho}$,

$$\langle \text{conv } v, \text{conv } v' \rangle \in \llbracket \alpha_l \rightarrow \mathbf{int} \rrbracket_{\rho}.$$

From the definition of conv , we need to prove that:

$$\langle v, v' \rangle \in \llbracket \alpha_l \rightarrow \mathbf{int} \rrbracket_{\rho}.$$

We have two cases:

- $l \sqsubseteq \zeta$. We have that $\rho(\alpha_l) = \rho(\alpha_l^f) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$ and $\rho(\alpha_l^f) = \langle \mathbf{int}, \mathbf{int}, \text{id}_{\mathbf{int}} \rangle$. We need to prove that $\langle v \langle \rangle, v' \langle \rangle \rangle \in \llbracket \alpha_l^f \rrbracket_{\rho}^{\text{ev}}$. Since $(v, v') \in \llbracket \alpha_l^f \rightarrow \alpha^f \rrbracket_{\rho}$, from $\rho(\alpha_l^f)$ and $\rho(\alpha^f)$ it follows that $v \langle \rangle = v' \langle \rangle$. Therefore, we have that $\langle v, v' \rangle \in \llbracket \alpha_l \rightarrow \mathbf{int} \rrbracket_{\rho}$.
- $l \not\sqsubseteq \zeta$. We have that $\rho(\alpha_l) = \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle$ and hence, $\langle v, v' \rangle \in \llbracket \alpha_l \rightarrow \mathbf{int} \rrbracket_{\rho}$ vacuously.

Lemma 41. For any ζ , for any α_l^f and α^f corresponding to x s.t. $lvl(x) = l$ and $\mathbf{F}(x) = \langle f, l' \rangle$:

$$\langle \text{dec}_f, \text{dec}_f \rangle \in I_{\mathcal{P}}^{\zeta} \llbracket (\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_{l'} \rightarrow \tau_f \rrbracket.$$

Proof. Let ρ be an environment s.t. $\rho \models_{\zeta} \mathcal{P}$. From Def. 15, we need to prove that:

$$\langle \text{dec}_f, \text{dec}_f \rangle \in \llbracket (\alpha_l^f \rightarrow \alpha^f) \rightarrow \alpha_{l'} \rightarrow \tau_f \rrbracket_{\rho}.$$

That is for any $\langle v_1, v_2 \rangle \in \llbracket \alpha_l^f \rightarrow \alpha^f \rrbracket_{\rho}$, $\langle \text{dec}_f v_1, \text{dec}_f v_2 \rangle \in \llbracket \alpha_{l'} \rightarrow \tau_f \rrbracket_{\rho}^{\text{ev}}$. From the definition of dec_f , we need to prove that

$$\langle \lambda_- : \mathbf{unit}.f(v_1 \langle \rangle), \lambda_- : \mathbf{unit}.f(v_2 \langle \rangle) \rangle \in \llbracket \alpha_{l'} \rightarrow \tau_f \rrbracket_{\rho}$$

When $l' \not\sqsubseteq \zeta$, we have that $\rho(\alpha_{l'}) = \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle$ and hence, the statement holds vacuously. We now consider the case $l' \sqsubseteq \zeta$. We have that $\rho(\alpha_{l'}) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$ and $\rho(\alpha_l^f) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$. Therefore we need to prove that

$$\langle f(v_1 \langle \rangle), f(v_2 \langle \rangle) \rangle \in \llbracket \tau_f \rrbracket_{\rho}^{\text{ev}}$$

Let $v'_i = v_i \langle \rangle$. We need to prove that

$$\langle f(v'_1), f(v'_2) \rangle \in \llbracket \tau_f \rrbracket_{\rho}^{\text{ev}}$$

Note that since $\langle v_1, v_2 \rangle \in \llbracket \alpha_l^f \rightarrow \alpha^f \rrbracket_{\rho}$ and $\rho(\alpha_l^f) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$, we have that

$$\langle v_1 \langle \rangle, v_2 \langle \rangle \rangle \in \llbracket \alpha^f \rrbracket_{\rho}^{\text{ev}}.$$

In other words, $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^f \rrbracket_{\rho}$.

We have two sub-cases:

- $l \not\sqsubseteq \zeta$. We have that $\rho(\alpha_l^f) = \langle \mathbf{int}, \mathbf{int}, R_f \rangle$. Since $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^f \rrbracket_{\rho} = R_f$, we have that $\langle f v'_1, f v'_2 \rangle \in \llbracket \tau_f \rrbracket_{\emptyset}^{\text{ev}}$. Therefore, $\langle f(v_1 \langle \rangle), f(v_2 \langle \rangle) \rangle \in \llbracket \tau_f \rrbracket_{\emptyset}^{\text{ev}}$ and hence, $\langle f(v_1 \langle \rangle), f(v_2 \langle \rangle) \rangle \in \llbracket \tau_f \rrbracket_{\rho}^{\text{ev}}$ (note that τ_f is a closed type).
- $l \sqsubseteq \zeta$. We have that $\rho(\alpha_l^f) = \langle \mathbf{int}, \mathbf{int}, id_{\mathbf{int}} \rangle$. Therefore $\langle v'_1, v'_2 \rangle \in id_{\mathbf{int}}$ and hence, $v'_1 = v'_2 = v$. Since $\vdash f v : \tau_f$, from Theorem 8, we have that $\langle f v, f v \rangle \in \llbracket \tau_f \rrbracket_{\emptyset}^{\text{ev}}$. Therefore, $\langle f(v_1 \langle \rangle), f(v_2 \langle \rangle) \rangle \in \llbracket \tau_f \rrbracket_{\emptyset}^{\text{ev}}$ and hence, $\langle f(v_1 \langle \rangle), f(v_2 \langle \rangle) \rangle \in \llbracket \tau_f \rrbracket_{\rho}^{\text{ev}}$ (note that τ_f is a closed type).

J.3 Proofs for extensions of the multi-level encoding

Proofs for the extension with multiple declassifiers We prove that the implementations of cp_- , cv_- , cvu_- , hi_{f_i} , and $mi1_{g_i}$ are indistinguishable to themselves for any observer.

Lemma 42. *For any ζ , it follows that:*

$$\begin{aligned}
\langle \text{comp}, \text{comp} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [\forall \beta_1, \beta_2. (\alpha_l \rightarrow \beta_1) \rightarrow (\beta_1 \rightarrow (\alpha_l \rightarrow \beta_2)) \rightarrow \alpha_l \rightarrow \beta_2] \\
\langle \text{convup}, \text{convup} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [\forall \beta. (\alpha_l \rightarrow \beta) \rightarrow (\alpha_{l'} \rightarrow \beta)] \\
\langle \text{wrap}, \text{wrap} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [\forall \beta. \beta \rightarrow \alpha_l \rightarrow \beta] \\
\langle \text{conv}, \text{conv} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [(\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_H \rightarrow \mathbf{int}] \\
\langle \text{conv}, \text{conv} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [(\alpha_{M_1}^{g_1, g_2} \rightarrow \alpha^{g_1, g_2}) \rightarrow \alpha_{M_1} \rightarrow \mathbf{int}]
\end{aligned}$$

Proof. The proofs for **comp**, **convup**, and **wrap** are as the corresponding ones in the proof of Lemma 38. We now prove the first part about **conv**. The proof for the second part about **conv** is similar.

Let ρ be an environment s.t. $\rho \models_\zeta^{\text{full}} \mathcal{P}_{M1}$. From the definition of indistinguishability, we need to prove that $\langle \text{conv}, \text{conv} \rangle \in [(\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_H \rightarrow \mathbf{int}]_\rho$. That is for any $(v, v') \in [\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}]_\rho$,

$$\langle \text{conv } v, \text{conv } v' \rangle \in [\alpha_H \rightarrow \mathbf{int}]_\rho.$$

From the definition of **conv**, we need to prove that:

$$\langle v, v' \rangle \in [\alpha_H \rightarrow \mathbf{int}]_\rho.$$

We have two cases:

- $H \sqsubseteq \zeta$. We have that $\rho(\alpha_H) = \rho(\alpha_H^{f_1, f_2}) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$ and $\rho(\alpha^{f_1, f_2}) = \langle \mathbf{int}, \mathbf{int}, \text{id}_{\mathbf{int}} \rangle$. We need to prove that $\langle v, v' \rangle \in [\mathbf{int}]_\rho^{\text{ev}}$. Since $(v, v') \in [\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}]_\rho$, from $\rho(\alpha_H)$ and $\rho(\alpha_H^{f_1, f_2})$, it follows that $v \langle \rangle = v' \langle \rangle$. Therefore, we have that $\langle v, v' \rangle \in [\alpha_H \rightarrow \mathbf{int}]_\rho$.
- $H \not\sqsubseteq \zeta$. We have that $\rho(\alpha_H) = \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle$ and hence, $\langle v, v' \rangle \in [\alpha_l \rightarrow \mathbf{int}]_\rho$ vacuously.

Lemma 43. *For any ζ ,*

$$\begin{aligned}
\langle \text{dec}_{f_1}, \text{dec}_{f_1} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [(\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_{M_1} \rightarrow \tau_{f_1}] \\
\langle \text{dec}_{f_2}, \text{dec}_{f_2} \rangle &\in \mathcal{I}_{\mathcal{P}_{M1}}^\zeta [(\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \alpha_{M_2} \rightarrow \tau_{f_2}]
\end{aligned}$$

Proof. We only prove the first part. The proof of the second part is similar.

Let ρ be an environment s.t. $\rho \models_\zeta^{\text{full}} \mathcal{P}_{M1}$. From the definition of indistinguishability, we need to prove that:

$$\langle \text{dec}_{f_1}, \text{dec}_{f_1} \rangle \in [(\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}) \rightarrow \alpha_{M_1} \rightarrow \tau_{f_1}]_\rho.$$

That is for any $\langle v_1, v_2 \rangle \in [\alpha_H^{f_1, f_2} \rightarrow \alpha^{f_1, f_2}]_\rho$, $\langle \text{dec}_{f_1} v_1, \text{dec}_{f_1} v_2 \rangle \in [\alpha_{M_1} \rightarrow \tau_{f_1}]_\rho^{\text{ev}}$. From the definition of **dec**_{*f*}, we need to prove that

$$\langle \lambda_- : \mathbf{unit}.f_1(v_1 \langle \rangle), \lambda_- : \mathbf{unit}.f_1(v_2 \langle \rangle) \rangle \in [\alpha_{M_1} \rightarrow \tau_{f_1}]_\rho$$

Let $v'_i = v_i \langle \rangle$. We need to prove that:

$$\langle \lambda_- : \mathbf{unit}.f_1(v'_1), \lambda_- : \mathbf{unit}.f_1(v'_2) \rangle \in \llbracket \alpha_{M_1} \rightarrow \tau_{f_1} \rrbracket_\rho$$

When $M_1 \not\sqsubseteq \zeta$, we have that $\rho(\alpha_{M_1}) = \langle \mathbf{unit}, \mathbf{unit}, \emptyset \rangle$ and hence, the statement holds vacuously. We now consider the case $M_1 \sqsubseteq \zeta$. We have that $\rho(\alpha_{M_1}) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$. Therefore we need to prove that

$$\langle f_1(v'_1), f_1(v'_2) \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$$

Note that since $M_1 \sqsubseteq \zeta$, we have that $\rho(\alpha^{f_1, f_2}_{M_1}) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$. Since $\langle v_1, v_2 \rangle \in \llbracket \alpha^{f_1, f_2}_H \rightarrow \alpha^{f_1, f_2} \rrbracket_\rho$, we have that $\langle v_1 \langle \rangle, v_2 \langle \rangle \rangle \in \llbracket \alpha^{f_1, f_2} \rrbracket_\rho^{\text{ev}}$. In other words, $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^{f_1, f_2} \rrbracket_\rho$.

We have two sub-cases:

- $\zeta = M_1$. We have that $\rho(\alpha^{f_1, f_2}_H) = \langle \mathbf{int}, \mathbf{int}, R_{f_1} \rangle$. Since $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^{f_1, f_2} \rrbracket_\rho = R_{f_1}$, we have that $\langle f_1 v'_1, f_1 v'_2 \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$ and hence, $\langle f_1 v'_1, f_1 v'_2 \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$ (note that τ_f is a closed type). Therefore, $\langle f_1(v_1 \langle \rangle), f_1(v_2 \langle \rangle) \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$.
- $\zeta = H$. We have that $\rho(\alpha^{f_1, f_2}_H) = \langle \mathbf{int}, \mathbf{int}, id_{\mathbf{int}} \rangle$. Therefore $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^{f_1, f_2} \rrbracket_\rho = id_{\mathbf{int}}$ and hence, $v'_1 = v'_2 = v$. Since $\vdash f_1 v : \tau_f$, from Theorem 8, we have that $\langle f_1 v, f_1 v \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$ and hence, $\langle f_1 v, f_1 v \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$ (note that τ_f is a closed type). Thus, $\langle f_1(v_1 \langle \rangle), f_1(v_2 \langle \rangle) \rangle \in \llbracket \tau_{f_1} \rrbracket_\rho^{\text{ev}}$.

Lemma 44. *For any ζ ,*

$$\begin{aligned} \langle \text{dec}_{g_1}, \text{dec}_{g_1} \rangle &\in \check{I}_{\mathcal{P}_{M_1}}^\zeta \llbracket (\alpha^{g_1, g_2}_{M_1} \rightarrow \alpha^{g_1, g_2}) \rightarrow \alpha_L \rightarrow \tau_{g_1} \rrbracket \\ \langle \text{dec}_{g_2}, \text{dec}_{g_2} \rangle &\in \check{I}_{\mathcal{P}_{M_1}}^\zeta \llbracket (\alpha^{g_1, g_2}_{M_1} \rightarrow \alpha^{g_1, g_2}) \rightarrow \alpha_L \rightarrow \tau_{g_2} \rrbracket \end{aligned}$$

Proof. We only prove the first part. The proof of the second part is similar.

Let ρ be an environment s.t. $\rho \models_\zeta^{\text{full}} \mathcal{P}_{M_1}$. From the definition of indistinguishability, we need to prove that:

$$\langle \text{dec}_{g_1}, \text{dec}_{g_1} \rangle \in \llbracket \alpha^{g_1, g_2}_{M_1} \rightarrow \alpha^{g_1, g_2} \rightarrow \alpha_L \rightarrow \tau_{g_1} \rrbracket_\rho.$$

That is for any $\langle v_1, v_2 \rangle \in \llbracket \alpha^{g_1, g_2}_{M_1} \rightarrow \alpha^{g_1, g_2} \rrbracket_\rho$, $\langle \text{dec}_{g_1} v_1, \text{dec}_{g_1} v_2 \rangle \in \llbracket \alpha_L \rightarrow \tau_{g_1} \rrbracket_\rho^{\text{ev}}$. From the definition of dec_{g_1} and the logical relation, we need to prove that

$$\langle \lambda_- : \mathbf{unit}.g_1(v_1 \langle \rangle), \lambda_- : \mathbf{unit}.g_1(v_2 \langle \rangle) \rangle \in \llbracket \alpha_L \rightarrow \tau_{g_1} \rrbracket_\rho$$

Let v'_i s.t. $v_i \langle \rangle \rightarrow^* v'_i$. We need to prove that:

$$\langle \lambda_- : \mathbf{unit}.g_1(v'_1), \lambda_- : \mathbf{unit}.g_1(v'_2) \rangle \in \llbracket \alpha_L \rightarrow \tau_{g_1} \rrbracket_\rho$$

For all ζ , we have that $L \sqsubseteq \zeta$ and hence, $\rho(\alpha_L) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$. Therefore we need to prove that

$$\langle g_1(v'_1), g_1(v'_2) \rangle \in \llbracket \tau_{g_1} \rrbracket_\rho^{\text{ev}}$$

Note that $\rho(\alpha^{g_1, g_2}_{M_1}) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$. Since $\langle v_1, v_2 \rangle \in \llbracket \alpha^{g_1, g_2}_{M_1} \rightarrow \alpha^{g_1, g_2} \rrbracket_\rho$, we have that $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^{g_1, g_2} \rrbracket_\rho$.

We have two sub-cases:

- $M_1 \not\sqsubseteq \zeta$ (i.e. $\zeta = L$ or $\zeta = M_2$). We have that $\rho(\alpha^{g_1, g_2}) = \langle \mathbf{int}, \mathbf{int}, R_{g_1, g_2} \rangle$. Since $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^{g_1, g_2} \rrbracket_\rho = R_{g_1, g_2}$, we have that $\langle g_1 v'_1, g_1 v'_2 \rangle \in \llbracket \tau_{g_1} \rrbracket_\rho^{\text{ev}}$ and hence, $\langle g_1 v'_1, g_1 v'_2 \rangle \in \llbracket \tau_{g_1} \rrbracket_\rho^{\text{ev}}$ (note that τ_{g_1} is a closed type).
- $M_1 \sqsubseteq \zeta$. We have that $\rho(\alpha^{g_1, g_2}) = \langle \mathbf{int}, \mathbf{int}, id_{\mathbf{int}} \rangle$. Therefore $\langle v'_1, v'_2 \rangle \in id_{\mathbf{int}}$ and hence, $v'_1 = v'_2 = v$. Since $\vdash g_1 v : \tau_f$, from Theorem 8, we have that $\langle g_1 v, g_1 v \rangle \in \llbracket \tau_{g_1} \rrbracket_\rho^{\text{ev}}$ and hence, $\langle g_1 v, g_1 v \rangle \in \llbracket \tau_{g_1} \rrbracket_\rho^{\text{ev}}$ (note that τ_{g_1} is a closed type).

Theorem 12. *If $\Delta_{\mathcal{P}}, \Gamma_{\mathcal{P}} \vdash e : \tau$, then for any $\zeta \in \mathcal{L}_\diamond$ and $\rho \models_\zeta^{\text{full}} \mathcal{P}_{M_1}$,*

$$\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_{\mathcal{P}_{M_1}}^\zeta \llbracket \tau \rrbracket^{\text{ev}}.$$

Proof. Since $\rho \models_\zeta^{\text{full}} \mathcal{P}$, from the definition of $\rho \models_\zeta^{\text{full}} \mathcal{P}_{M_1}$, Lemma 42, Lemma 43, and Lemma 44, it follows that $\rho \models_\zeta \Delta_{\mathcal{P}_{M_1}}, \Gamma_{\mathcal{P}_{M_1}}$. Since $\Delta_{\mathcal{P}_{M_1}}, \Gamma_{\mathcal{P}_{M_1}} \vdash e : \tau$, from Theorem 8, we have that $\langle \rho_L(e), \rho_R(e) \rangle \in \llbracket \tau \rrbracket_\rho^{\text{ev}}$. From the definition of indistinguishability, it follows that $\langle \rho_L(e), \rho_R(e) \rangle \in \mathcal{I}_{\mathcal{P}_{M_1}}^\zeta \llbracket \tau \rrbracket^{\text{ev}}$.

Proofs for the extension for global policies The proof of the main result for this extension (in Section I.4) is similar to the one in Section I.4. Here, we only prove that dec_f relates to itself.

Lemma 45. *For any ζ ,*

$$\langle \text{dec}_f, \text{dec}_f \rangle \in \mathcal{I}_{\mathcal{P}_{Av}}^\zeta \llbracket (\alpha_{M_1, M_2}^f \rightarrow \alpha^f) \rightarrow \alpha_L \rightarrow \mathbf{int} \rrbracket$$

Proof. Let ρ be an environment s.t. $\rho \models_\zeta^{\text{full}} \mathcal{P}_{Av}$. From the definition of indistinguishability, we need to prove that:

$$\langle \text{dec}_{f_1}, \text{dec}_{f_1} \rangle \in \llbracket \alpha_{M_1, M_2}^f \rightarrow \alpha^f \rrbracket_\rho \rightarrow \alpha_L \rightarrow \mathbf{int} \rrbracket_\rho.$$

That is for any $\langle v_1, v_2 \rangle \in \llbracket \alpha_{M_1, M_2}^f \rightarrow \alpha^f \rrbracket_\rho$, $\langle \text{dec}_f v_1, \text{dec}_f v_2 \rangle \in \llbracket \alpha_L \rightarrow \mathbf{int} \rrbracket_\rho^{\text{ev}}$. From the definition of dec_f , we need to prove that

$$\langle \lambda_- : \mathbf{unit}.f(v_1 \mathbf{unit}), \lambda_- : \mathbf{unit}.f(v_2 \mathbf{unit}) \rangle \in \llbracket \alpha_L \rightarrow \mathbf{int} \rrbracket_\rho$$

Let v'_i be s.t. $v_i \langle \rangle \rightarrow^* v'_i$. We need to prove that:

$$\langle \lambda_- : \mathbf{unit}.f(v'_1), \lambda_- : \mathbf{unit}.f(v'_2) \rangle \in \llbracket \alpha_L \rightarrow \mathbf{int} \rrbracket_\rho$$

Since L is the smallest element in the lattice, for all ζ , $\rho(\alpha_L) = \text{Full}_{\mathbf{unit}}$. Thus, we need to prove that $f v'_1 = f v'_2$.

Note that $\rho(\alpha_{M_1, M_2}^f) = \langle \mathbf{unit}, \mathbf{unit}, \text{Full}_{\mathbf{unit}} \rangle$. Since $\langle v_1, v_2 \rangle \in \llbracket \alpha_{M_1, M_2}^f \rightarrow \alpha^f \rrbracket_\rho$, we have that $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^f \rrbracket_\rho$.

- $\zeta = H$. We have that $\rho(\alpha^f) = \langle \mathbf{int} \times \mathbf{int}, \mathbf{int} \times \mathbf{int}, id_{\mathbf{int} \times \mathbf{int}} \rangle$. Therefore, we have that $v'_1 = v'_2$ and hence, $f v'_1 = f v'_2$.
- $\zeta \in \{M_1, M_2, L\}$. We have that $\rho(\alpha_f) = \langle \mathbf{int} \times \mathbf{int}, \mathbf{int} \times \mathbf{int}, R_f^\bullet \rangle$, where

$$R_f^\bullet = \{ \langle v_1, v_2 \rangle \mid \langle f v_1, f v_2 \rangle \in \llbracket \mathbf{int} \rrbracket_\rho^{\text{ev}} \}$$

Since $\langle v'_1, v'_2 \rangle \in \llbracket \alpha^f \rrbracket_\rho = R_f^\bullet$, from the definition of $\llbracket \mathbf{int} \rrbracket_\rho^{\text{ev}}$ we have that $f v'_1 = f v'_2$.