Linear Declassification

Yûta Kaneko Naoki Kobayashi

Tohoku University
{kaneko,koba}@kb.ecei.tohoku.ac.jp

Abstract. We propose a new notion of declassification policy called *linear declassification*. Linear declassification controls not only which functions may be applied to declassify high-security values, but also *how often* the declassification functions may be applied. We present a linear type system which guarantees that well-typed programs never violate linear declassification policies. To state a formal security property guaranteed by the linear declassification, we also introduce *linear relaxed non-interference* as an extension of Li and Zdancewic's relaxed non-interference. An application of the linear relaxed non-interference to quantitative information flow analysis is also discussed.

1 Introduction

There have been extensive studies on policies and verification methods for information flow security [4, 6, 9, 10, 12, 14]. The standard policy for secure information flow is the *non-interference* property, which means that low-security outputs cannot be affected by high-security inputs. A little more formally, a program e is secure if for any high inputs h_1 and h_2 and low input l, $e(h_1, l)$ and $e(h_2, l)$ are equivalent for low-level observers. The standard non-interference property is, however, too restricted in practice, since it does not allow any leakage of secret information. For example, a login program does leak information about the result of comparison of a string and a password.

To allow intentional release of secret information, a variety of notions of declassification have been proposed [6, 11, 12]. Sabelfeld and Myers [11] proposed delimited information release, where e is secure if, roughly speaking, whenever $d(h_1) = d(h_2)$ for the declassification function d, $e(h_1, l)$ and $e(h_2, l)$ are equivalent for low-level observers. As a similar criterion, Li and Zdancewic [6] proposed a notion of relaxed non-interference, where e is secure (i.e., satisfies relaxed non-interference) if $e(h, l)$ is factorized into $e'(dh)$, where d is a declassification function and e' does not contain h. Both the frameworks guarantee that a program leaks only partial information $d(h)$ about the high-security value h. For example, if d is the function $\lambda x.x \bmod 2$, then only the parity information can be leaked.

The above criteria alone, however, do not always guarantee desirable secrecy properties. For example, consider a declassification function $d \stackrel{\triangle}{=} \lambda x.\lambda s.(s = x)$, which takes a high-security value x, and returns a *function* that takes a string and returns whether s and x are equal. Declassifications through such a function often occur in practice, for instance, in a login program, which compares a user's

password with an input string. Note that $d(h) \stackrel{\triangle}{=} \lambda s.(s = h)$ and h contain the same quantity of information; In fact, h can be factorized into:

$$(\lambda g.\textbf{let } test(s) = \textbf{if } g(s) \textbf{ then } s \textbf{ else } test(s+1) \textbf{ in } test(0))\,(d(h)).$$

Thus, the above criteria guarantee nothing about the quantity of information declassified through the function d.

To overcome the problem mentioned above, we propose a new notion of declassification called *linear declassification*, which controls *how often* declassification functions can be applied to each high-security value, and how often a value (which may be a function) obtained by declassification may be used. We define a linear type system that ensures that any well-typed program satisfies a given linear declassification policy.

To formalize the security property guaranteed by the linear declassification, we also extend Li and Zdancewic's relaxed non-interference [6] to *linear relaxed non-interference*, which says that e is secure if e can be factorized into $e'\lambda^u x.(dh)$, where e' does not contain h *and* e' *can call the function* $\lambda x.(dh)$ *at most* u *times to declassify the value of* h. The linear relaxed non-interference is useful for quantitative analysis of information flow [2, 3, 7]. For example, if a program e containing an n-bit password satisfies the linear relaxed non-interference under the policy that $\lambda x.\lambda s.(x = s)$ is used at most once, we know that one has to run e $O(2^n)$ times in average to get complete information about the password. On the other hand, if the declassification function is replaced by $\lambda x.\lambda s.(s > x)$, the password may be leaked by only n runs of the program.

The rest of this paper is structured as follows. Section 2 introduces the language of programs and linear declassification policies. Section 3 introduces a linear type system which guarantees that a program adheres to linear declassification policies. Section 4 defines linear relaxed non-interference as an extension of Li and Zdancewic's relaxed non-interference. Section 4 also discusses how linear relaxed non-interference can be used for quantitative analysis of information flow. Section 5 discusses related work and Section 6 concludes.

2 Language

This section introduces the syntax and semantics of programs and declassification policies.

2.1 Syntax

Definition 1 (expressions) The set of *expressions*, ranged over by e, is defined by:

$$\begin{aligned} e \text{ (expressions)} ::= {} & x \mid n \mid \sigma \mid d\langle\!\langle e \rangle\!\rangle \mid e_1 \oplus e_2 \mid \textbf{if } e_1 \textbf{ then } e_2 \textbf{ else } e_3 \\ & \mid \lambda^u x.e \mid \textbf{fix } x(y) = e \mid e_1 e_2 \mid \langle e_1, \ldots, e_n \rangle \mid \#_i(e) \\ u \text{ (uses)} ::= {} & 0 \mid 1 \mid \omega \\ \oplus \text{ (operators)} ::= {} & + \mid - \mid = \mid \cdots \end{aligned}$$

Here, the meta-variables x and n range over the sets of variables and integers respectively. The meta-variable σ ranges over the set of special integers, to which secrecy policies (given below) are associated. For the sake of simplicity, we consider only integers as primitive values, and assume that $e_1 = e_2$ returns 1 if the values of e_1 and e_2 are the same, and returns 0 otherwise. **if** e_1 **then** e_2 **else** e_3 returns the value of e_3 if the value of e_1 is 0, and returns the value of e_2 otherwise. The expression $\lambda^u x.e$ denotes a function that can be used at most u times. If u is ω, the function can be used an arbitrary number of times.[1] The expression **fix** $x(y) = e$ denotes a recursive function that can be used an arbitrary number of times. The expression $e_1 e_2$ is an ordinary function application. The expression $d\langle\!\langle e \rangle\!\rangle$ is a special form of function application, where the meta-variable d ranges over the set $\mathcal{N}_D$ of special function variables (defined in a policy introduced below). The expression $\langle e_1, \ldots, e_n \rangle$ returns a tuple consisting of the values of $e_1, \ldots, e_n$. Note that n may be 0, in which case, the tuple is empty.

We write $[e'/x]e$ for the (capture-avoiding) substitution of e' for x in e. We write $\mathbf{SVar}(e)$ for the set of security variables occurring in e.

Definition 2 (policies) The set of *policies* is defined by:

$$\begin{aligned} p \text{ (security levels)} &::= \mathbf{L} \mid \mathbf{H} \mid \{d_1 \mapsto u_1, \cdots, d_n \mapsto u_n\} \\ D \text{ (declassification environment)} &::= \{d_1 \mapsto \lambda^\omega x.e_1, \cdots, d_n \mapsto \lambda^\omega x.e_2\} \\ \Sigma \text{ (policy)} &::= \{\sigma_1 \mapsto p_1, \cdots, \sigma_n \mapsto p_n\} \end{aligned}$$

A security level p expresses the degree of confidentiality of each value. If p is $\mathbf{L}$, the value may be leaked to low-security principals. If p is $\mathbf{H}$, no information about the value may be leaked. If p is $\{d_1 \mapsto u_1, \cdots, d_n \mapsto u_n\}$, then the value may be leaked only through declassification functions $d_1, \ldots, d_n$ and each declassification function d_i may be applied to the value at most u_i times. For example, if the security level of σ is $\{d_1 \mapsto 1, d_2 \mapsto \omega, d_3 \mapsto 0\}$, then $d_1\langle\!\langle\sigma\rangle\!\rangle + d_2\langle\!\langle\sigma\rangle\!\rangle + d_2\langle\!\langle\sigma\rangle\!\rangle$ is allowed, but neither $d_3\langle\!\langle\sigma\rangle\!\rangle$ nor $d_1\langle\!\langle\sigma\rangle\!\rangle + d_1\langle\!\langle\sigma\rangle\!\rangle$ is.

A declassification environment D defines declassification functions. A policy Σ maps σ_i to its security level. Note that the use of $D(d_i)$ is always ω. This is because how often d_i can be used is described in Σ for each security variable σ.

Example 1. Let $D = \{d \mapsto \lambda^\omega x.\lambda^1 y.x = y\}$ and $\Sigma = \{\sigma \mapsto \{d \mapsto 1\}\}$. This policy specifies that information about σ can be leaked by at most one application of d. Since the result of the application is a linear (use-once) function $\lambda^1 y.\sigma = y$, the policy means that σ may be compared with another integer only once.

Note that if $D(d)$ is $\lambda^\omega x.\lambda^\omega y.x = y$, then the declassification may be performed only once, but the resulting value $\lambda^\omega y.\sigma = y$ can be used an arbitrary number of times. Therefore, an attacker can obtain complete information about σ by applying the function to different values.

[1] For the sake of simplicity, we consider only $0, 1, \omega$ as uses. It is easy to extend the language and the type system given in the next section to allow $2, 3, \ldots$.

2.2 Operational Semantics

This section introduces an operational semantics to define the meaning of expressions and policies formally.

A run-time state is modeled by a pair $\langle H, e\rangle$, where H is a heap given below.[2]

Definition 3 (heap)

$$\begin{aligned} H \text{ (heap)} ::= \{&f_1 \mapsto \lambda^{u_1} x_1.e_1, \ldots, f_n \mapsto \lambda^{u_n} x_n.e_n, \\ &\sigma_1 \mapsto (n_1, p_1), \ldots, \sigma_m \mapsto (n_m, p_m)\} \\ f \text{ (function pointer)} ::= &\ x \mid d \end{aligned}$$

Here, f ranges over the set consisting of (ordinary) variables $(x, y, z, \ldots)$ and declassification function variables $(d_1, d_2, \ldots,)$.

A heap H keeps information about how often each function may be applied and how the value of each security variable may be declassified in the rest of the computation. For example, $H(\sigma) = (2, \{d \mapsto 1\})$ means that the value of σ is 2, and the value can be declassified only once through the declassification function d.

For a system (Σ, D, e), the initial heap is determined by Σ, D, and the actual values of the security variables. Let g be a mapping from $dom(\Sigma)$ to the set of integers. We write $H_{\Sigma,D,g}$ for the initial heap $D \cup \{\sigma_1 \mapsto (g(\sigma_1), \Sigma(\sigma_1)), \ldots, \sigma_k \mapsto (g(\sigma_k), \Sigma(\sigma_k))\}$ (where $dom(\Sigma) = \{\sigma_1, \ldots, \sigma_k\}$). We use evaluation contexts to define the transition relation.

Definition 4 (evaluation context) The set of evaluation contexts, ranged over by E, is given by:

$$\begin{aligned} E \text{ (evaluation context)} ::= &[\,] \mid [\,]e \mid x[\,] \mid d\langle\langle[\,]\rangle\rangle \mid \textbf{if } [\,] \textbf{ then } e_1 \textbf{ else } e_2 \\ &\mid [\,] \oplus e \mid v \oplus [\,] \mid \langle v_1, \ldots, v_{k-1}, [\,], e_{k+1}, \ldots, e_n\rangle \mid \#_i([\,]) \\ v \text{ (values)} ::= &\ f \mid n \mid \sigma \mid \langle v_1, \ldots, v_n\rangle \end{aligned}$$

The relation $\langle H, e\rangle \longrightarrow \langle H', e'\rangle$ is the least relation closed under the rules in Figure 1. In the figure, $F\{x \mapsto v\}$ is the mapping F' such that $F'(x) = v$, and $F'(y) = F(y)$ for any $y \in dom(F) \setminus \{x\}$. $val(H, v)$ is defined to be n if $v = n$, or $v = \sigma$ and $H(\sigma) = (n, p)$.

The key rules are E-App and E-Declassify. In E-App, the use of the function y is decreased by one. Here, the subtraction $u-1$ is defined by: $1-1 = 0$ and $\omega - 1 = \omega$. Note that $0-1$ is undefined, so that if $H(y) = \lambda^0 x.e$, the function y can no longer be used (in other words, the evaluation of $E[yv]$ get stuck).

In E-Declassify, the security level p for σ changes after the reduction. Here, $p - d$ is defined by:

$$\begin{aligned} &\{d_1 \mapsto u_1, \ldots, d_n \mapsto u_n\} - d_i = \{d_1 \mapsto u'_1, \ldots, d_n \mapsto u'_n\} \\ &\qquad \text{where } u'_j = \begin{cases} u_j - 1 & \text{if } j = i \\ u_j & \text{otherwise} \end{cases} \\ &\mathbf{L} - d_i = \mathbf{L} \end{aligned}$$

[2] Note that unlike the usual heap-based semantics, tuples are *not* stored in a heap.

For example, if the security level p of σ is $\{d \mapsto 1\}$, then after the declassification, the security level becomes $p - d = \{d \mapsto 0\}$, which means that the value of σ can no longer be declassified. Note that $\mathbf{H} - d_i$ is undefined, so that an integer of security level $\mathbf{H}$ can never be declassified. Rule E-DECLASSIFY2 is for the case when a declassification function d is applied to an ordinary integer.

In rule E-OP, $\underline{\oplus}$ is the binary operation on integers denoted by the operator symbol $\oplus$. The remaining rules are standard.

$$\frac{y \text{ fresh}}{\langle H, E[\lambda^u x.e]\rangle \longrightarrow \langle H\{y \mapsto \lambda^u x.e\}, E[y]\rangle} \quad \text{(E-FUN)}$$

$$\frac{H(d) = \lambda^\omega x.e}{\langle H\{\sigma \mapsto (n,p)\}, E[d\langle\!\langle\sigma\rangle\!\rangle]\rangle \longrightarrow \langle H\{\sigma \mapsto (n, p-d)\}, E[[n/x]e]\rangle} \quad \text{(E-DECLASSIFY)}$$

$$\frac{H(d) = \lambda^\omega x.e \qquad v \notin dom(\Sigma)}{\langle H, E[d\langle\!\langle n\rangle\!\rangle]\rangle \longrightarrow \langle H, E[[n/x]e]\rangle} \quad \text{(E-DECLASSIFY2)}$$

$$\langle H\{y \mapsto \lambda^u x.e\}, E[yv]\rangle \longrightarrow \langle H\{y \mapsto \lambda^{u-1} x.e\}, E[[v/x]e]\rangle \quad \text{(E-APP)}$$

$$\frac{val(H, v) \neq 0}{\langle H, E[\textbf{if } v \textbf{ then } e_1 \textbf{ else } e_2]\rangle \longrightarrow \langle H, E[e_1]\rangle} \quad \text{(E-IFT)}$$

$$\frac{val(H, v) = 0}{\langle H, E[\textbf{if } v \textbf{ then } e_1 \textbf{ else } e_2]\rangle \longrightarrow \langle H, E[e_2]\rangle} \quad \text{(E-IFF)}$$

$$\langle H, E[v_1 \oplus v_2]\rangle \longrightarrow \langle H, E[val(H, v_1) \underline{\oplus} val(H, v_2)]\rangle \quad \text{(E-OP)}$$

$$\frac{z \text{ fresh}}{\langle H, E[\textbf{fix } x(y) = e]\rangle \longrightarrow \langle H \cup \{z \mapsto \lambda^\omega y.[z/x]e\}, E[z]\rangle} \quad \text{(E-FIX)}$$

$$\langle H, E[\#_i\langle v_1, \ldots, v_n\rangle]\rangle \longrightarrow \langle H, E[v_i]\rangle \quad \text{(E-PROJ)}$$

Fig. 1. Evaluation rules

Example 2. Recall the security policy in Example 1: $D = \{d \mapsto \lambda^\omega x.\lambda^1 y.(x = y)\}$ and $\Sigma = \{\sigma \mapsto \{d \mapsto 1\}\}$.
$\langle H_{\Sigma,D,\{\sigma\mapsto 3\}}, d\langle\!\langle\sigma\rangle\!\rangle 2\rangle$ is reduced as follows.

$$\begin{aligned}
&\langle D \cup \{\sigma \mapsto (3, \{d \mapsto 1\})\}, d\langle\!\langle\sigma\rangle\!\rangle 2\rangle \\
\longrightarrow\ &\langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\})\}, (\lambda^1 y.(3 = y))2\rangle \\
\longrightarrow\ &\langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^1 y.(3 = y)\}, z(2)\rangle \\
\longrightarrow\ &\langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^0 y.(3 = y)\}, 3 = 2\rangle \\
\longrightarrow\ &\langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^0 y.(3 = y)\}, 0\rangle
\end{aligned}$$

On the other hand, both $\langle d\langle\!\langle\sigma\rangle\!\rangle, d\langle\!\langle\sigma\rangle\!\rangle\rangle$ and $(\lambda^{\omega} f.\langle f(1), f(2)\rangle)(d\langle\!\langle\sigma\rangle\!\rangle)$ get stuck as follows.

$$\begin{array}{l} \quad \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 1\})\}, \langle d\langle\!\langle\sigma\rangle\!\rangle, d\langle\!\langle\sigma\rangle\!\rangle\rangle\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\})\}, \langle \lambda^{1} y.(3 = y), d\langle\!\langle\sigma\rangle\!\rangle\rangle\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^{1} y.(3 = y)\}, \langle z, d\langle\!\langle\sigma\rangle\!\rangle\rangle\rangle \\ \not\longrightarrow \end{array}$$

$$\begin{array}{l} \quad \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 1\})\}, (\lambda^{\omega} f.\langle f(1), f(2)\rangle)(d\langle\!\langle\sigma\rangle\!\rangle)\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 1\}), z \mapsto \lambda^{\omega} f.\langle f(1), f(2)\rangle\}, z(d\langle\!\langle\sigma\rangle\!\rangle)\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^{\omega} f.\langle f(1), f(2)\rangle\}, z(\lambda^{1} y.(3 = y))\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^{\omega} f.\langle f(1), f(2)\rangle, w \mapsto \lambda^{1} y.(3 = y)\}, z(w)\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^{\omega} f.\langle f(1), f(2)\rangle, w \mapsto \lambda^{1} y.(3 = y)\}, \langle w(1), w(2)\rangle\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^{\omega} f.\langle f(1), f(2)\rangle, w \mapsto \lambda^{0} y.(3 = y)\}, \langle 3 = 1, w(2)\rangle\rangle \\ \longrightarrow \langle D \cup \{\sigma \mapsto (3, \{d \mapsto 0\}), z \mapsto \lambda^{\omega} f.\langle f(1), f(2)\rangle, w \mapsto \lambda^{0} y.(3 = y)\}, \langle 0, w(2)\rangle\rangle \\ \not\longrightarrow \end{array}$$

□

3 Type System

This section introduces a linear type system, which ensures that if $\langle \Sigma, D, e\rangle$ is well-typed, then e satisfies the security policy specified by Σ and D.

3.1 Types

Definition 5 (types) The set of types, ranged over by τ, is defined by:

$$\begin{array}{l} \tau \text{ (types)} ::= \mathit{int}_p \mid \tau_1 \xrightarrow{\varphi}_u \tau_2 \mid \langle \tau_1, \ldots, \tau_n \rangle \\ \varphi \text{ (effects)} ::= \mathtt{t} \mid \mathtt{nt} \end{array}$$

The integer type int_p describes integers whose security level is p. For example, $\mathit{int}_{\{d \mapsto 1\}}$ is the type of integers that can be declassified through the function d at most once. The function type $\tau_1 \xrightarrow{\varphi}_u \tau_2$ describes functions that can be used at most u times and that take a value of type τ_1 as an argument and return a value of type τ_2. The effect φ describes whether the function is terminating (when $\varphi = \mathtt{t}$) or it may not be terminating (when $\varphi = \mathtt{nt}$). The effect will be used for preventing leakage of information from the termination behavior of a program. The type $\langle \tau_1, \ldots, \tau_n \rangle$ describes tuples consisting of values of types $\tau_1, \ldots, \tau_n$.

The *sub-effect relation* $\leq$ on effects is the partial order defined by $\mathtt{t} \leq \mathtt{nt}$. The *sub-level relation* $\sqsubseteq$ on security levels and the subtyping relation $\tau_1 \leq \tau_2$ are the least relations closed under the rules in Figure 2. For example, $\mathit{int}_{\{d \mapsto 1\}} \xrightarrow{\mathtt{t}}_{\omega} \mathit{int}_{\{d \mapsto \omega\}}$ is a subtype of $\mathit{int}_{\{d \mapsto \omega\}} \xrightarrow{\mathtt{nt}}_{1} \mathit{int}_{\{d \mapsto 1\}}$. We write $\varphi_1 \vee \varphi_2$ for the least upper bound of φ_1 and φ_2 (with respect to $\leq$), and $p_1 \sqcup p_2$ for the least upper bound of p_1 and p_2 with respect to $\sqsubseteq$.

$$\mathbf{L} \sqsubseteq p \sqsubseteq \mathbf{H} \qquad \text{(PSub1)}$$

$$\frac{u'_i \leq u_i \text{ for each } i \in \{1, \ldots, m\}}{\{d_1 \mapsto u_1, \ldots, d_m \mapsto u_m, \ldots\} \sqsubseteq \{d_1 \mapsto u'_1, \ldots, d_m \mapsto u'_m\}} \qquad \text{(PSub2)}$$

$$\frac{p_1 \sqsubseteq p_2}{int_{p_1} \leq int_{p_2}} \qquad \text{(S-Policies)}$$

$$\frac{\tau'_1 \leq \tau_1 \qquad \tau_2 \leq \tau'_2 \qquad u' \leq u \qquad \varphi \leq \varphi'}{\tau_1 \xrightarrow{\varphi}_u \tau_2 \leq \tau'_1 \xrightarrow{\varphi'}_{u'} \tau'_2} \qquad \text{(S-Fun)}$$

Fig. 2. Subtyping rules

3.2 Typing

A type environment is a mapping from a finite set consisting of extended variables (ordinary variables, security variables, and declassification function names) to types. We have two forms of type judgment: $\vdash \langle \Sigma, D, e \rangle$ for the whole system (consisting of a policy, a declassification environment, and an expression), and $\Gamma \vdash e : \tau \,\&\, \varphi$ for expressions. The judgment $\vdash \langle \Sigma, D, e \rangle$ means that e satisfies the security policy specified by Σ and D. $\Gamma \vdash e : \tau \,\&\, \varphi$ means that e evaluates to a value of type τ under an environment described by Γ. If $\varphi = \mathtt{t}$, then evaluation of e must terminate. If $\varphi = \mathtt{nt}$, then e may or may not terminate. For example, $\sigma : int_{\{d \mapsto 1\}}, f : int_{\{d \mapsto 1\}} \xrightarrow{\mathtt{t}}_{\omega} int_{\{d \mapsto 1\}} \vdash f\sigma : int_{\{d \mapsto 1\}} \,\&\, \mathtt{t}$ is a valid judgment, but neither $\sigma : int_{\{d \mapsto 1\}}, f : int_{\{d \mapsto \omega\}} \xrightarrow{\mathtt{t}}_{\omega} int_{\{d \mapsto 1\}} \vdash f\sigma : int_{\{d \mapsto 1\}} \,\&\, \mathtt{t}$ nor $\sigma : int_{\{d \mapsto 1\}}, f : int_{\{d \mapsto 1\}} \xrightarrow{\mathtt{nt}}_{\omega} int_{\{d \mapsto 1\}} \vdash f\sigma : int_{\{d \mapsto 1\}} \,\&\, \mathtt{t}$ is. (In the former, the security level of σ does not match that of the argument required by f. In the latter, the type of f says that f may not terminate, but the conclusion says that $f\sigma$ terminates.)

Figure 3 shows the typing rules. Two auxiliary judgments $\vdash \Sigma : \Gamma$ and $\vdash D : \Gamma$ are used for defining $\vdash \langle \Sigma, D, e \rangle$. The definitions of the operations used in the typing rules are summarized in Figure 4.

We explain some key rules below.

- T-Op: Suppose e_1 has type $int_{\{d \mapsto 1\}}$. Then, the value of e_1 can be declassified through the function d, but that does not necessarily imply that $e_1 \oplus e_2$ can be declassified through the function d. Therefore, we raise the security level of $e_1 \oplus e_2$ to $\mathbf{H}$ unless both of the security levels of e_1 and e_2 are $\mathbf{L}$.
- T-If: Since information about the value of e_0 indirectly flows to the value of the if-expression, the security level of the if-expression should be greater than or equal to the *ceil* of security level of e_0. For the sake of simplicity, we require that the values of if-expressions must be integers.
- T-Fun: The premise means that free variables are used according to Γ *each time* the function is applied. Since the function may be applied u times, the usage of free variables is expressed by $u \cdot \Gamma$ in total.
- T-Dcl: The premise ensures that e must have type $int_{d \mapsto 1}$, so that e can indeed be declassified through d.

Example 3. Let $\tau_d = int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{\omega} int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{1} int_{\mathbf{L}}$. $d\langle\!\langle\sigma\rangle\!\rangle 2$ is typed as follows.

$$\dfrac{\dfrac{\sigma : int_{\{d\mapsto 1\}} \vdash \sigma : int_{\{d\mapsto 1\}} \,\&\, \mathtt{t}}{d:\tau_d, \sigma : int_{\{d\mapsto 1\}} \vdash d\langle\!\langle\sigma\rangle\!\rangle : int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{1} int_{\mathbf{L}} \,\&\, \mathtt{t}} \quad \emptyset \vdash 2 : int_{\mathbf{L}} \,\&\, \mathtt{t}}{d:\tau_d, \sigma : int_{\{d\mapsto 1\}} \vdash d\langle\!\langle\sigma\rangle\!\rangle 2 : int_{\mathbf{L}} \,\&\, \mathtt{t}}$$

Example 4. Let e be $\mathbf{fix}\ f(x) = \mathbf{if}\ d\langle\!\langle\sigma\rangle\!\rangle x\ \mathbf{then}\ x\ \mathbf{else}\ f(x+1)$. It is typed as follows:

$$\dfrac{\dfrac{\Gamma_2 \vdash d\langle\!\langle\sigma\rangle\!\rangle x : int_L \,\&\, \mathtt{t} \quad \Gamma_3 \vdash x : int_L \,\&\, \mathtt{t} \quad \Gamma_3 \vdash f(x+1) : int_L \,\&\, \mathtt{nt}}{\Gamma_2, f : int_{\mathbf{L}} \xrightarrow{\mathtt{nt}}_{\omega} int_{\mathbf{L}}, \vdash \mathbf{if}\ d\langle\!\langle\sigma\rangle\!\rangle x\ \mathbf{then}\ x\ \mathbf{else}\ f(x+1) : int_{\mathbf{L}} \,\&\, \mathtt{nt}}}{\Gamma_1 \vdash e : int_{\mathbf{L}} \,\&\, \mathtt{nt}}$$

Here, Γ_1, Γ_2, and Γ_3 are:

$$\begin{aligned}
\Gamma_1 &= d : int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{\omega} int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{1} int_{\mathbf{L}}, \sigma : int_{\{d\mapsto\omega\}} \\
\Gamma_2 &= d : int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{1} int_{\mathbf{L}} \xrightarrow{\mathtt{t}}_{1} int_{\mathbf{L}}, \sigma : int_{\{d\mapsto 1\}}, x : int_{\mathbf{L}} \\
\Gamma_3 &= f : int_{\mathbf{L}} \xrightarrow{\mathtt{nt}}_{\omega} int_{\mathbf{L}}, x : int_{\mathbf{L}}
\end{aligned}$$

Let $\Sigma_1 = \{\sigma \mapsto \{d \mapsto \omega\}\}$, $\Sigma_2 = \{\sigma \mapsto \{d \mapsto 1\}\}$, and $D = \{d \mapsto \lambda^{\omega} x.\lambda^{1} y.(x = y)\}$. Then, $\vdash \langle \Sigma_1, D, e(0)\rangle : int_{\mathbf{L}}$ holds but $\vdash \langle \Sigma_2, D, e(0)\rangle : int_{\mathbf{L}}$ does not.

3.3 (Partial) Type Soundness

The following theorem means that evaluation of a well-typed program never gets stuck. A proof is given in Appendix B.

Theorem 1. *Suppose that* $dom(\Sigma) = \{\sigma_1, \ldots, \sigma_k\}$.
If $\vdash \langle \Sigma, D, e\rangle$ *and* $\langle H_{\Sigma,D,\{\sigma_1\mapsto n_1,\ldots,\sigma_k\mapsto n_k\}}, e\rangle \longrightarrow^{*} \langle H, e'\rangle \not\longrightarrow$, *then* e' *is a value.*

Note that Theorem 1 only guarantees that evaluation does not get stuck because of invalid usage of declassification functions; The theorem alone does not necessarily guarantee that e satisfies the security policy. In fact, the evaluation of $\langle H_{\emptyset,\emptyset,\{\sigma\mapsto 2\}}, \sigma + 1\rangle$ does not get stuck (yields the value 3), but it does leak information about σ. The security property satisfied by well-typed programs is formalized in the next section.

4 Linear Relaxed Non-Interference

In this section, we define *linear relaxed non-interference* as a new criterion of information flow security, and prove that well-typed programs of our type system satisfy that criterion. Linear relaxed non-interference is an extension of relaxed non-interference [6]. We first review relaxed non-interference and discuss its weakness in Section 4.1. We then define linear relaxed non-interference in Section 4.2. Section 4.3 shows that any programs well-typed in our type system satisfy the linear relaxed non-interference. Section 4.4 discusses an application of linear relaxed non-interference to quantitative information flow analysis.

$\Gamma \vdash e : \tau$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau \,\&\, \mathtt{t}} \quad \text{(T-Var)}$$

$$\frac{}{\Gamma \vdash n : int_{\mathbf{L}} \,\&\, \mathtt{t}} \quad \text{(T-Const)}$$

$$\frac{}{\Gamma, \sigma : int_p \vdash \sigma : int_p \,\&\, \mathtt{t}} \quad \text{(T-SVal)}$$

$$\frac{\Gamma_1 \vdash e_1 : int_{p_1} \,\&\, \varphi \qquad \Gamma_2 \vdash e_2 : int_{p_2} \,\&\, \varphi}{\Gamma_1 + \Gamma_2 \vdash e_1 \oplus e_2 : int_{\lceil p_1 \rceil \sqcup \lceil p_2 \rceil} \,\&\, \varphi} \quad \text{(T-Op)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \,\&\, \varphi}{u \cdot \Gamma \vdash \lambda^u x.e : \tau_1 \xrightarrow{\varphi}_u \tau_2 \,\&\, \mathtt{t}} \quad \text{(T-Fun)}$$

$$\frac{\Gamma, x : \tau_1 \xrightarrow{\mathtt{nt}}_\omega \tau_2, y : \tau_1 \vdash e : \tau_2 \,\&\, \varphi}{\omega \cdot \Gamma \vdash \mathbf{fix}\ x(y) = e : \tau_1 \xrightarrow{\varphi}_\omega \tau_2 \,\&\, \mathtt{t}} \quad \text{(T-Fix)}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \xrightarrow{\varphi_0}_1 \tau_2 \,\&\, \varphi_1 \qquad \Gamma_2 \vdash e_2 : \tau_1 \,\&\, \varphi_2}{\Gamma_1 + \Gamma_2 \vdash e_1\ e_2 : \tau_2 \,\&\, \varphi_0 \vee \varphi_1 \vee \varphi_2} \quad \text{(T-App)}$$

$$\frac{\Gamma \vdash e : int_{\{d \mapsto 1\}} \,\&\, \varphi_1}{(d : int_{\mathbf{L}} \xrightarrow{\varphi_0}_\omega \tau) + \Gamma \vdash d\langle\!\langle e \rangle\!\rangle : \tau \,\&\, \varphi_0 \vee \varphi_1} \quad \text{(T-Dcl)}$$

$$\frac{\Gamma \vdash e : \tau' \,\&\, \varphi' \qquad \tau' \leq \tau \qquad \varphi' \leq \varphi}{\Gamma \vdash e : \tau \,\&\, \varphi} \quad \text{(T-Sub)}$$

$$\frac{\Gamma_1 \vdash e_0 : int_{p_0} \,\&\, \varphi_0 \qquad \Gamma_2 \vdash e_1 : int_{p_1} \,\&\, \varphi_1 \qquad \Gamma_2 \vdash e_2 : int_{p_2} \,\&\, \varphi_2 \qquad \varphi_1 = \varphi_2 = \mathtt{t} \text{ if } \lceil p_0 \rceil = \mathbf{H}}{\Gamma_1 + \Gamma_2 \vdash \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : int_{\lceil p \rceil \sqcup p_1 \sqcup p_2} \,\&\, \varphi_0 \vee \varphi_1 \vee \varphi_2} \quad \text{(T-If)}$$

$$\frac{\Gamma_i \vdash e_i : \tau_i \,\&\, \varphi_i \text{ (for each } i \in \{1, \ldots, n\})}{\Gamma_1 + \cdots + \Gamma_n \vdash \langle e_1, \ldots, e_n \rangle : \langle \tau_1, \ldots, \tau_n \rangle \,\&\, \varphi_1 \vee \cdots \vee \varphi_n} \quad \text{(T-Tuple)}$$

$\vdash \Sigma : \Gamma$

$$\frac{}{\vdash \{\sigma_1 \mapsto p_1, \ldots, \sigma_n \mapsto p_n\} : (\sigma_1 : int_{p_1}, \ldots, \sigma_n : int_{p_n})} \quad \text{(T-Policy)}$$

$\vdash D : \Gamma$

$$\frac{\emptyset \vdash \lambda^\omega x.e_i : \tau_i \,\&\, \varphi_i \text{ for each } i \in \{1, \ldots, n\}}{\vdash \{d_1 \mapsto \lambda^\omega x.e_1, \cdots, d_n \mapsto \lambda^\omega x.e_n\} : (d_1 : \tau_1, \ldots, d_n : \tau_n)} \quad \text{(T-DEnv)}$$

$\vdash \langle \Sigma, D, e \rangle$

$$\frac{\vdash \Sigma : \Gamma_1 \qquad \vdash D : \Gamma_2 \qquad \Gamma_1, \Gamma_2 \vdash e : \tau \,\&\, \varphi \qquad \text{all the security levels in } \Gamma_2 \text{ are } \mathbf{L}}{\vdash \langle \Sigma, D, e \rangle : \tau} \quad \text{(T-Sys)}$$

Fig. 3. Typing rules

$$u_1 + u_2 = \begin{cases} 0 \text{ if } u_1 = u_2 = 0 \\ 1 \text{ if } (u_1, u_2) \in \{(0,1),(1,0)\} \\ \omega \text{ otherwise} \end{cases}$$

$$int_{\mathbf{L}} + int_{\mathbf{L}} = int_{\mathbf{L}} \qquad int_{\mathbf{H}} + int_{\mathbf{H}} = int_{\mathbf{H}}$$

$$int_{\{d_1 \mapsto u_1, \ldots, d_n \mapsto u_n\}} + int_{\{d_1 \mapsto u'_1, \ldots, d_n \mapsto u'_n\}} = int_{\{d_1 \mapsto (u_1 + u'_1), \ldots, d_n \mapsto (u_n + u'_n)\}}$$

$$(\tau_1 \xrightarrow{\varphi}_u \tau_2) + (\tau_1 \xrightarrow{\varphi}_{u'} \tau_2) = \tau_1 \xrightarrow{\varphi}_{(u+u')} \tau_2$$

$$(\Gamma_1 + \Gamma_2)\ (x) = \begin{cases} \Gamma_1\ (x) & \text{if } x \in dom(\Gamma_1) \setminus dom(\Gamma_2) \\ \Gamma_2\ (x) & \text{if } x \in dom(\Gamma_2) \setminus dom(\Gamma_1) \\ \Gamma_1\ (x) + \Gamma_2\ (x) & \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \end{cases}$$

$$u_1 \cdot u_2 = \begin{cases} 0 \text{ if } u_1 = 0 \text{ or } u_2 = 0 \\ 1 \text{ if } u_1 = u_2 = 1 \\ \omega \text{ otherwise} \end{cases}$$

$$u \cdot int_L = int_L \qquad u \cdot int_H = int_H$$

$$u \cdot int_{\{d_1 \mapsto u_1, \ldots, d_n \mapsto u_n\}} = int_{\{d_1 \mapsto u \cdot u_1, \ldots, d_n \mapsto u \cdot u_n\}}$$

$$u \cdot (\tau_1 \to_{u'} \tau_2) = \tau_1 \to_{u \cdot u'} \tau_2$$

$$(u \cdot \Gamma)\ (x) = u \cdot \Gamma(x)$$

$$\lceil p \rceil = \begin{cases} \mathbf{L} \text{ if } p = \mathbf{L} \\ \mathbf{H} \text{ otherwise} \end{cases}$$

Fig. 4. Operations on policies, types, and type environments

4.1 Relaxed Non-Interference

Relaxed non-interference [6] is an extension of non-interference. Suppose that $\Sigma = \{\sigma \mapsto \{d \mapsto \omega\}\}$. Informally, an expression e satisfies relaxed non-interference under the policy Σ if e can be factorized (up to a certain program equivalence) into $e'(d\sigma)$, where e' does not contain σ. If d is a constant function $\lambda x.0$, then the relaxed non-interference degenerates into the standard non-interference.

As already discussed in Section 1, the relaxed non-interference does not always guarantee a desired secrecy property. For example, consider the case where $d = \lambda x.\lambda y.x = y$. Then, *any* expression containing σ can be factorized into $e'(d\sigma)$ up to the standard contextual equivalence. In fact, σ is contextually-equivalent to:[3]

$$(\lambda^{\omega} g.(\mathbf{fix}\ test(s) = \mathbf{if}\ g(s)\ \mathbf{then}\ s\ \mathbf{else}\ test(s+1))\ 0)(d\langle\!\langle\sigma\rangle\!\rangle)$$

4.2 Linear Relaxed Non-Interference

We first define the notion of (typed) contextual equivalence. For the sake of simplicity, we consider only closed terms (thus, it suffices to consider only contexts of the form $e[\,]$). We write $\langle H, e\rangle \Downarrow n$ if $\langle H, e\rangle \longrightarrow^* \langle H', n\rangle$ for some n.

[3] Actually, Li and Zdancewic [6] uses a finer equivalence than the contextual equivalence, so that the above factorization is not valid. However, if σ ranges over a finite set, then a similar factorization is possible by unfolding the recursion.

Definition 6 (contextual equivalence) Suppose that $\emptyset \vdash e_1 : \tau \,\&\, \varphi$ and $\emptyset \vdash e_2 : \tau \,\&\, \varphi$. e_1 and e_2 are *contextually equivalent*, written $e_1 \approx_{\tau,\varphi} e_2$, if, for any e such that $\emptyset \vdash e : \tau \xrightarrow{\mathtt{nt}}_{\omega} \mathit{int}_{\mathbf{L}}$, $\langle \emptyset, ee_1 \rangle \Downarrow 0$ if and only if $\langle \emptyset, ee_2 \rangle \Downarrow 0$.

We now define the linear relaxed non-interference.

Definition 7 (Linear Relaxed Non-interference) Let $\Sigma = \{\sigma_1 \mapsto \{d_1 \mapsto u_{11}, \ldots, d_k \mapsto u_{1k}\}, \ldots, \sigma_m \mapsto \{d_1 \mapsto u_{m1}, \ldots, d_k \mapsto u_{mk}\}\}$. Suppose also that $\mathbf{SVar}(e) \subseteq \{\sigma_1, \ldots, \sigma_m\}$. $\langle \Sigma, D, e \rangle$ satisfies *linear relaxed non-interference* at τ if there exists e' such that the following equivalence holds for any integers $n_1, \ldots, n_m$:

$$[n_1/\sigma_1, \ldots, n_m/\sigma_m]D(e) \approx_{\tau,\mathtt{nt}} e' \; \langle \lambda^{u_{11}} x.(D(d_1)n_1), \ldots, \lambda^{u_{1k}} x.(D(d_k)n_1) \rangle \\ \cdots \\ \langle \lambda^{u_{m1}} x.(D(d_1)n_m), \ldots, \lambda^{u_{mk}} x.(D(d_k)n_m) \rangle$$

Here $D(e)$ denotes the term obtained from e by replacing each occurrence of a declassification expression $d\langle\!\langle e \rangle\!\rangle$ with $D(d)e$.

Intuitively, the above definition means that if $\langle \Sigma, D, e \rangle$ satisfies *linear relaxed non-interference*, then e can leak information about the security variables $\sigma_1, \ldots, \sigma_m$ only by calling declassification functions at most the number of times specified by Σ. Note that in the above definition, e' cannot depend on the values of the security variables $n_1, \ldots, n_m$.

4.3 Soundness of the Type System

We now show that well-typed programs satisfy linear relaxed non-interference.

Theorem 2. *If $\vdash \langle \Sigma, D, e \rangle : \tau$ and all the security levels in τ are $\mathbf{L}$, then $\langle \Sigma, D, e \rangle$ satisfies the linear relaxed non-interference at τ.*

A proof of the above theorem is given in Appendix C. We explain below an outline of the proof. We shall introduce a transformation relation $\Gamma \vdash e : \tau \rightsquigarrow e'$, which should be read "the term e that has type τ under Γ is transformed into the term e'." By the transformation, an integer n of type $\mathit{int}_{\{d_1:u_1,\ldots,d_m,u_m\}}$ is replaced by a tuple $\langle \lambda^{u_1} x.(d_1\langle\!\langle n \rangle\!\rangle), \ldots, \lambda^{u_m} x.(d_m\langle\!\langle n \rangle\!\rangle) \rangle$, which consists of functions for declassifying n. Thus, declassification $d_k\langle\!\langle e \rangle\!\rangle$ is replaced by a projection $\#_k(e')\langle\,\rangle$, where e' is the term obtained by transforming e. On the other hand, a high-security integer n of type $\mathit{int}_{\mathbf{H}}$ is replaced by the unit value $\langle\,\rangle$. For example, $(\lambda^{\omega} x : \mathit{int}_{\{d_1 \mapsto u_1,\ldots,d_m \mapsto u_m\}}.d_j\langle\!\langle x \rangle\!\rangle)n$ is transformed into:[4]

$$(\lambda^{\omega} x.\#_j(x)\langle\,\rangle)\langle \lambda^{u_1} x_1.(d_1\langle\!\langle n \rangle\!\rangle), \ldots, \lambda^{u_m} x_m.(d_m\langle\!\langle n \rangle\!\rangle) \rangle.$$

The key features of the transformation are that the transformation preserves the semantics of programs (i.e., a term of type $\mathit{int}_{\mathbf{L}}$ is transformed into a term that

[4] Here, we annotated x with its type since the actual transformation depends on the type of x.

evaluates to the same integer), and that an integer n of type $int_{\{d_1:u_1,\ldots,d_m,u_m\}}$ can appear only in the form $(\lambda^{u_1}x.(d_1\langle\langle n\rangle\rangle),\ldots,\lambda^{u_m}x.(d_m\langle\langle n\rangle\rangle))$ after the transformation.

Given the transformation relation, the theorem can be proved as follows. Suppose $\vdash \langle\Sigma, D, e\rangle : \tau$. Then there exist Γ_1, Γ_2 such that $\vdash \Sigma : \Gamma_1$, $\vdash D : \Gamma_2$, and $\Gamma_1, \Gamma_2 \vdash e : \tau \,\&\, \varphi$. Let e'' be a term such that

$$[x_1/\sigma,\ldots,x_k/\sigma_k]\Gamma_1, \Gamma_2 \vdash [x_1/\sigma,\ldots,x_k/\sigma_k]e : \tau \,\&\, \varphi \rightsquigarrow e''.$$

Then, the condition on the linear relaxed non-interference is satisfied for $e' = \lambda^1 x_1.\cdots\lambda^1 x_k.D(e'')$.

4.4 Application to Quantitative Information Flow Analysis

In this subsection, we discuss how linear relaxed non-interference can be applied to quantitative information flow analysis [2, 7]. Unlike the classical information flow analysis, which obtains *binary* information of whether or not a high-security value is leaked to public, the quantitative analysis aims to estimate the *quantity* of the information leakage based. Recently, definitions and methods of the quantitative information flow analysis have been extensively studied by Malacaria et al. [2, 7], based on Shannon's information theory [13]. The quantitative analysis is generally more expensive than the classical information flow analysis, and has not been fully automated. As discussed below, the linear relaxed non-interference helps us reduce the cost of the quantitative analysis.

For the sake of simplicity, we consider below only a single high security variable σ and the declassification environment $D = \{d \mapsto \lambda^\omega x.\lambda^1 y.x \oplus y\}$, with the fixed security policy $\Sigma = \{\sigma \mapsto \{d \mapsto 1\}\}$.

Suppose that $\langle\Sigma, D, e\rangle$ satisfies linear relaxed non-interference at $int_{\mathbf{L}}$. Let us consider the *quantity* of information that flows from σ to the value of e. By Definition 7, there exists an e' such that for any n and n_1, $\langle\{\sigma \mapsto (n,p)\} \cup D, e\rangle \Downarrow n_1$ if and only if $\langle\{\sigma \mapsto (n,p)\} \cup D, e'\langle\lambda^1 x.d\langle\langle\sigma\rangle\rangle\rangle\rangle \Downarrow n_1$, where e' does not contain σ. Moreover, since $e'(\lambda^1 x.d\langle\langle\sigma\rangle\rangle)$ is well-typed, if $\langle\{\sigma \mapsto (n,p)\} \cup D, e'\langle\lambda^1 x.d\langle\langle\sigma\rangle\rangle\rangle\rangle \longrightarrow^* \langle H, n_1\rangle$ and the value of σ is used during the reduction, then the reduction se-

quence must be of the following form:[5]

$$
\begin{array}{rl}
& \langle \{\sigma \mapsto (n, \{d \mapsto 1\})\} \cup D, e'\langle \lambda^1 x.d\langle\!\langle \sigma \rangle\!\rangle \rangle \rangle \\
\longrightarrow^* & \langle \{\sigma \mapsto (n, \{d \mapsto 1\})\} \cup H_1, E_1[\lambda^1 x.d\langle\!\langle \sigma \rangle\!\rangle] \rangle \\
\longrightarrow & \langle \{\sigma \mapsto (n, \{d \mapsto 1\}), z \mapsto \lambda^1 x.d\langle\!\langle \sigma \rangle\!\rangle\} \cup H_1, E_1[z] \rangle \\
\longrightarrow^* & \langle \{\sigma \mapsto (n, \{d \mapsto 1\}), z \mapsto \lambda^1 x.d\langle\!\langle \sigma \rangle\!\rangle\} \cup H_2, E_2[z\langle\,\rangle] \rangle \\
\longrightarrow & \langle \{\sigma \mapsto (n, \{d \mapsto 1\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle\} \cup H_2, E_2[d\langle\!\langle \sigma \rangle\!\rangle] \rangle \\
\longrightarrow & \langle \{\sigma \mapsto (n, \{d \mapsto 0\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle\} \cup H_2, E_2[\lambda^1 y.n \oplus m] \rangle \\
\longrightarrow & \langle \{\sigma \mapsto (n, \{d \mapsto 0\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle, w \mapsto \lambda^1 y.n \oplus y\} \cup H_2, E_2[w] \rangle \\
\longrightarrow^* & \langle \{\sigma \mapsto (n, \{d \mapsto 0\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle, w \mapsto \lambda^1 y.n \oplus y\} \cup H_3, E_3[w(m)] \rangle \\
\longrightarrow & \langle \{\sigma \mapsto (n, \{d \mapsto 0\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle, w \mapsto \lambda^0 y.n \oplus y\} \cup H_3, E_3[n \oplus m] \rangle \\
\longrightarrow & \langle \{\sigma \mapsto (n, \{d \mapsto 0\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle, w \mapsto \lambda^0 y.n \oplus y\} \cup H_3, E_3[m'] \rangle \\
\longrightarrow^* & \langle \{\sigma \mapsto (n, \{d \mapsto 0\}), z \mapsto \lambda^0 x.d\langle\!\langle \sigma \rangle\!\rangle, w \mapsto \lambda^0 y.n \oplus y\} \cup H_4, n_1 \rangle
\end{array}
$$

Here, since e' does not contain σ, H_i and E_i $(i = 1, 2, 3)$ are independent of the value n of σ.

Let $\mathtt{L}$ be a random variable representing e' above, $\mathtt{H}$ be a random variable representing the value n of σ, and $\mathtt{O}$ be a random variable representing the final value n_1. Then, by the reduction sequence above, $\mathtt{O}$ can be expressed as follows.

$$\mathtt{O} = f_0(f_1(\mathtt{L}), \mathtt{H} \underline{\oplus} f_2(\mathtt{L}))$$

Here, $f_1(\mathtt{L})$ corresponds to the pair (H_3, E_3) and $f_2(\mathtt{L})$ corresponds to m in the reduction step above. The function f_0 represents the computation of n_1 from the configuration $\langle \{\sigma \mapsto (n, \{d \mapsto 0\}), \ldots\} \cup H_3, E_3[m'] \rangle$.

According to [2, 7], the leakage of information is expressed by:[6]

$$\mathcal{I}(\mathtt{O}; \mathtt{H} \mid \mathtt{L}) = \mathcal{H}(\mathtt{O} \mid \mathtt{L}) = \mathcal{H}(\mathtt{O}, \mathtt{L}) - \mathcal{H}(\mathtt{L})$$

Here, $\mathcal{H}(\vec{X})$ is defined as $\Sigma_x P(\vec{X} = \vec{x}) \log \frac{1}{P(\vec{X} = \vec{x})}$ (and $P(\vec{X} = \vec{x})$ denotes the probability that the value of $\vec{X}$ is $\vec{x}$).

Using $\mathtt{O} = f_0(f_1(\mathtt{L}), \mathtt{H} \underline{\oplus} f_2(\mathtt{L}))$, $\mathcal{I}(\mathtt{O}; \mathtt{H} \mid \mathtt{L})$ is estimated as follows.

$$
\begin{array}{rll}
\mathcal{I}(\mathtt{O}; \mathtt{H} \mid \mathtt{L}) & = \mathcal{H}(\mathtt{O}, \mathtt{L}) - \mathcal{H}(\mathtt{L}) & \\
& = \mathcal{H}(f_0(f_1(\mathtt{L}), \mathtt{H} \underline{\oplus} f_2(\mathtt{L})), \mathtt{L}) - \mathcal{H}(\mathtt{L}) & \\
& \leq \mathcal{H}(f_1(\mathtt{L}), \mathtt{H} \underline{\oplus} f_2(\mathtt{L}), \mathtt{L}) - \mathcal{H}(\mathtt{L}) & \text{(by Appendix A, Lemma 1)} \\
& = \mathcal{H}(\mathtt{H} \underline{\oplus} f_2(\mathtt{L}), \mathtt{L}) - \mathcal{H}(\mathtt{L}) & \text{(by the definition of } \mathcal{H}) \\
& = \mathcal{H}(\mathtt{H} \underline{\oplus} f_2(\mathtt{L}) \mid \mathtt{L}) & \text{(by the definition of } \mathcal{H}(\mathtt{X} \mid \mathtt{Y})) \\
& \leq \mathcal{H}(\mathtt{H} \underline{\oplus} f_2(\mathtt{L}) \mid f_2(\mathtt{L})) & \text{(by Appendix A, Lemma 2)}
\end{array}
$$

Thus, $\mathcal{I}(\mathtt{O}; \mathtt{H} \mid \mathtt{L})$ is bound by the maximum information leakage by the operation $\underline{\oplus}$ (more precisely, the maximum value of $\mathcal{H}(\mathtt{H} \underline{\oplus} \mathtt{X} \mid \mathtt{X})$ obtained by changing the distribution for $\mathtt{X}$).

---

[5] For the sake of simplicity, we consider only terminating programs. Non-terminating programs can be treated in a similar manner, by introducing a special value $\perp$ for representing non-termination.

[6] Note that we are considering deterministic programs. Note also that we do not consider timing attacks. It is possible to hide timing attacks to some extent, by using Agat's technique, for instance [1].

If $\underline{\oplus}$ is the equality test for k-bit integers, then

$$\begin{aligned}
\mathcal{H}(\mathtt{H}\underline{\oplus}\mathtt{X} \mid \mathtt{X}) &= P(\mathtt{H} = \mathtt{X}) \log \tfrac{1}{P(\mathtt{H}=\mathtt{X})} + P(\mathtt{H} \neq \mathtt{X}) \log \tfrac{1}{P(\mathtt{H}\neq\mathtt{X})} \\
&= \tfrac{1}{2^k} \log 2^k + \tfrac{2^k-1}{2^k} \log \tfrac{2^k}{2^k-1} \\
&= \tfrac{1}{2^k} \log 2^k + \tfrac{2^k-1}{2^k} \log (1 + \tfrac{1}{2^k-1}) \\
&\leq \tfrac{k}{2^k} + \tfrac{2^k-1}{2^k} \cdot \tfrac{1}{2^k-1} \qquad (\text{by } \log(1+x) \leq x) \\
&= \tfrac{k+1}{2^k}
\end{aligned}$$

Thus, the maximum leakage is bound by $\frac{k+1}{2^k}$ (which is considered safe if k is sufficiently large).

On the other hand, if $\underline{\oplus}$ is the inequality test $<$, then, the maximum value of $\mathcal{H}(\mathtt{H}\underline{\oplus}\mathtt{X} \mid \mathtt{X})$ is obtained by letting $P(\mathtt{X} = 2^{k-1}) = 1$.

$$\begin{aligned}
\mathcal{H}(\mathtt{H}\underline{\oplus}\mathtt{X} \mid \mathtt{X}) &= P(\mathtt{H} < 2^{k-1}) \log \tfrac{1}{P(\mathtt{H}<2^{k-1})} + P(\mathtt{H} \geq 2^{k-1}) \log \tfrac{1}{P(\mathtt{H}\geq 2^{k-1})} \\
&= \tfrac{1}{2} \log 2 + \tfrac{1}{2} \log 2 \\
&= 1
\end{aligned}$$

Thus, we know that 1 bit of information about σ may be leaked by each run of the program.

5 Related Work

There have been many studies on information flow security and declassification policies: see [10, 12] for a general survey and comparison of declassification policies. Most closely related to our work is Sabelfeld and Myers' work on delimited information release [11], and Li and Zdancewic's work on relaxed non-interference [6]. They control what functions can be used for declassification, but not *how often* the declassification functions may be used; thus, the relaxed non-interference alone is not sufficient for bounding the quantity of information leakage. Li and Zdancewic [6] allow more flexible declassification than ours; for example, if a declassification function for σ is $\lambda x.((x+1) = 2)$, then declassification can be performed in two steps, by first applying $\lambda x.x+1$ and then $\lambda y.y = 2$. We think it is possible to extend our linear type system to allow such flexible declassification, but we did not do so in this paper for the sake of simplicity.

Quantitative analysis of information flow has been recently studied by Malacaria et al. [2, 3, 7] for imperative languages. As demonstrated in Section 4.4, the linear relaxed non-interference allows us to apply quantitative analysis only to declassification functions instead of the whole program, by which enabling a combination of traditional information flow analysis (with linearity analysis) and quantitative information flow analysis. A limitation of our approach is that only $0, 1, \omega$ uses are considered, so that if a declassification is performed inside a recursive function, the number of declassifications is always estimated as ω. To remove that limitation, we need to generalize uses, possibly using dependent types (for example, we can write $\Pi n : int_{\mathbf{L}}.int_{\{d \mapsto n\}} \to int_{\mathbf{L}}$ for the type of functions that

takes an integer n and a high-security value x, and applies the declassification function d to x, n times).

Our type system can be considered an instance of linear type systems [5, 8, 15]. In the usual linear type systems, the type of an integer is annotated with how often the integer is accessed. In our type system, the type of an integer is annotated with how often each declassification function may be applied to the integer. We did not discuss a type inference algorithm in this paper, but a type inference algorithm (that is quadratic in the program size, provided that the number of declassification functions is constant) can be developed in a standard manner [8].

6 Conclusion

We introduced a new notion of declassification called *linear declassification*, which not only controls what functions can be used for declassifying high-security values but also *how often* the declassification functions may be applied. We have also introduced *linear relaxed non-interference* to formalize the property guaranteed by linear declassification. The linear relaxed non-interference enables integration of traditional type-based information flow analysis and quantitative information flow analysis, by allowing us to apply quantitative analysis locally to declassification functions.

References

1. J. Agat. Transforming out timing leaks. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 40–53, 2000.
2. D. Clark, S. Hunt, and P. Malacaria. Quantitative information flow, relations and polymorphic types. *Journal of Logic and Computation*, 15(2):181–199, 2005.
3. D. Clark, S. Hunt, and P. Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, 2007.
4. D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
5. N. Kobayashi. Quasi-linear types. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 29–42, 1999.
6. P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 158–170, 2005.
7. P. Malacaria. Assessing security threats of looping constructs. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 225–235, 2007.
8. T. Mogensen. Types for 0, 1 or many uses. In *Implementation of Functional Languages*, volume 1467 of *Lecture Notes in Computer Science*, pages 112–122, 1998.

9. F. Pottier and V. Simonet. Information flow inference for ML. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 319–330, 2002.
10. A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE J. Selected Areas in Communications*, 21(1):5–19, Jan. 2003.
11. A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Software Security - Theories and Systems, Second Mext-NSF-JSPS International Symposium (ISSS 2003)*, volume 3233 of *Lecture Notes in Computer Science*, pages 174–191. Springer-Verlag, 2003.
12. A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *18th IEEE Computer Security Foundations Workshop (CSFW-18 2005)*, pages 255–269. IEEE Computer Society Press, 2005.
13. C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.
14. G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages*, pages 355–364, 1998.
15. D. N. Turner, P. Wadler, and C. Mossin. Once upon a type. In *Proceedings of Functional Programming Languages and Computer Architecture*, pages 1–11, San Diego, California, 1995.

Appendix

A Properties of the entropy

Here, we prepare two lemmas about the entropy $\mathcal{H}(X)$. We write f, g for functions, and X, Y for random variables. $f(X)$ denotes the distribution of $f(x)$, when x changes according to the distribution of X.
(So, $\mathcal{H}(f(X)) = \Sigma_{a \in codom(f)} P(f(X) = a) \log \frac{1}{P(f(X)=a)}$.)

Lemma 1. $\mathcal{H}(f(X)) \leq \mathcal{H}(X)$.

Proof.

$$\begin{aligned}
&\mathcal{H}(X) - \mathcal{H}(f(X)) \\
&= \Sigma_x P(X = x) \log \tfrac{1}{P(X=x)} - \Sigma_{a \in codom(f)} P(f(X) = a) \log \tfrac{1}{P(f(X)=a)} \\
&= \Sigma_{a \in codom(f)} ((\Sigma_{x \in \{y \mid f(y)=a\}} P(X = x) \log \tfrac{1}{P(X=x)}) - P(f(X) = a) \log \tfrac{1}{P(f(X)=a)})
\end{aligned}$$

Let $\{y \mid f(y) = a\}$ be $\{x_1, \ldots, x_k\}$ and p_i be $P(X = x_i)$. Then,

$$\begin{aligned}
& ((\Sigma_{x \in \{y \mid f(y)=a\}} P(X = x) \log \tfrac{1}{P(X=x)}) - P(f(X) = a) \log \tfrac{1}{P(f(X)=a)}) \\
= \;& (p_1 \log \tfrac{1}{p_1} + \cdots + p_k \log \tfrac{1}{p_k}) - (p_1 + \cdots + p_k) \log \tfrac{1}{p_1 + \cdots + p_k} \\
= \;& p_1 (\log \tfrac{1}{p_1} - \log \tfrac{1}{p_1 + \cdots + p_k}) + \cdots + p_k (\log \tfrac{1}{p_k} - \log \tfrac{1}{p_1 + \cdots + p_k}) \\
= \;& p_1 \log \tfrac{p_1 + \cdots + p_k}{p_1} + \cdots + p_k \log \tfrac{p_1 + \cdots + p_k}{p_k} \\
\geq \;& 0
\end{aligned}$$

(Notice that $\frac{p_1 + \cdots + p_k}{p_i} \geq 1$, so that $p_i \log \frac{p_1 + \cdots + p_k}{p_i} \geq 0$.) Thus, we obtain $\mathcal{H}(X) - \mathcal{H}(f(X)) \geq 0$ as required. □

Lemma 2. $\mathcal{H}(g(Y, f(X)) \mid X) \leq \mathcal{H}(g(Y, f(X)) \mid f(X))$.

Proof. By the definition of $\mathcal{H}(A \mid B)$, we have:

$$\begin{aligned}
& \mathcal{H}(g(Y, f(X)) \mid f(X)) - \mathcal{H}(g(Y, f(X)) \mid X) \\
&= (\mathcal{H}(g(Y, f(X)), f(X)) - \mathcal{H}(f(X))) - (\mathcal{H}(g(Y, f(X)), X) - \mathcal{H}(X)) \\
&= (\mathcal{H}(X) - \mathcal{H}(f(X))) - (\mathcal{H}(g(Y, f(X)), X) - \mathcal{H}(g(Y, f(X)), f(X)))
\end{aligned}$$

Let $q_{a,b}$ be $P(g(Y, a) = b)$. Then, $\mathcal{H}(g(Y, f(X)), X) - \mathcal{H}(g(Y, f(X)), f(X))$ is estimated as follows.

$$\begin{aligned}
&\mathcal{H}(g(Y, f(X)), X) - \mathcal{H}(g(Y, f(X)), f(X)) \\
&= \Sigma_{x,b} P(g(Y, f(x)) = b \wedge X = x) \log \tfrac{1}{P(g(Y, f(x)) = b \wedge X = x)} \\
&\quad - \Sigma_{a,b} P(g(Y, a) = b \wedge f(X) = a) \log \tfrac{1}{P(g(Y, a) = b \wedge f(X) = a)} \\
&= \Sigma_{a,b} ((\Sigma_{x \in \{y \mid f(y)=a\}} P(g(Y, f(a)) = b \wedge X = x) \log \tfrac{1}{P(g(Y, a) = b \wedge X = x)}) \\
&\qquad - P(g(Y, a) = b \wedge f(X) = a) \log \tfrac{1}{P(g(Y, a) = b \wedge f(X) = a)}) \\
&= \Sigma_{a,b} q_{a,b} ((\Sigma_{x \in \{y \mid f(y)=a\}} P(X = x) \log \tfrac{1}{q_{a,b} P(X = x)}) - P(f(X) = a) \log \tfrac{1}{q_{a,b} P(f(X) = a)}) \\
&= \Sigma_{a,b} q_{a,b} ((\Sigma_{x \in \{y \mid f(y)=a\}} P(X = x) \log \tfrac{1}{P(X = x)}) - P(f(X) = a) \log \tfrac{1}{P(f(X) = a)}) \\
&= \Sigma_a (\Sigma_b q_{a,b}) ((\Sigma_{x \in \{y \mid f(y)=a\}} P(X = x) \log \tfrac{1}{P(X = x)}) - P(f(X) = a) \log \tfrac{1}{P(f(X) = a)}) \\
&\leq \Sigma_a ((\Sigma_{x \in \{y \mid f(y)=a\}} P(X = x) \log \tfrac{1}{P(X = x)}) - P(f(X) = a) \log \tfrac{1}{P(f(X) = a)}) \\
&= \mathcal{H}(X) - \mathcal{H}(f(X))
\end{aligned}$$

Thus, we have $\mathcal{H}(g(Y, f(X)) \mid f(X)) - \mathcal{H}(g(Y, f(X)) \mid X) \geq 0$ as required.

B Proof of Theorem 1

We first define a type judgment $\vdash_R \langle H, e\rangle$ for run-time states. We write $\Gamma_1 \leq \Gamma_2$ when $dom(\Gamma_2) \subseteq dom(\Gamma_1)$ and $\Gamma_1(x) \leq \Gamma_2(x)$ for each $x \in dom(\Gamma_2)$.

Definition 8 We define $\vdash_R H : \Gamma$ by:

$$\frac{\begin{array}{c}\Gamma = f_1 : \tau_1, \ldots, f_k : \tau_k, \sigma_1 : int_{p'_1}, \ldots, \sigma_m : int_{p'_m} \\ \Gamma_i \vdash \lambda^{u_i} x_i.e_i : \tau'_i \,\&\, \varphi_i (\text{for each } i \in \{1, \ldots, k\}) \\ f_1 : \tau'_1, \ldots, f_k : \tau'_k, \sigma_1 : int_{p_1}, \ldots, \sigma_m : int_{p_m} \leq \Gamma + \Gamma_1 + \cdots + \Gamma_k\end{array}}{\vdash_R \{f_1 \mapsto \lambda^{u_1} x_1.e_1, \ldots, f_k \mapsto \lambda^{u_k} x_k.e_k, \sigma_1 \mapsto (n_1, p_1), \ldots, \sigma_m \mapsto (n_m, p_m)\} : \Gamma}$$

We write $\vdash_R \langle H, e\rangle : \tau$ when $\vdash_R H : \Gamma$ and $\Gamma \vdash e : \tau \,\&\, \varphi$ for some Γ and φ.

Theorem 1 follows from the three lemmas below. Lemma 3 says that if a system $\langle \Sigma, D, e\rangle$ is well-typed, then the initial run-time state $\langle H_{\Sigma,D,g}, e\rangle$ is also well-typed. Lemma 4 says that if a run-time state is well-typed, it does not get stuck immediately. Lemma 5 says that well-typedness of a run-time state is preserved by reductions.

Lemma 3. *Let* $g = \{\sigma_1 \mapsto n_1, \ldots, \sigma_m \mapsto n_m\}$ *and* $dom(\Sigma) = \{\sigma_1, \ldots, \sigma_m\}$. *If* $\vdash \langle \Sigma, D, e\rangle : \tau$, *then* $\vdash_R \langle H_{\Sigma,D,g}, e\rangle : \tau$.

Proof. Suppose $\vdash \langle \Sigma, D, e\rangle : \tau$. Then, there exist Γ'_1 and Γ'_2 such that $\vdash \Sigma : \Gamma'_1$ and $\vdash D : \Gamma'_2$ with $\Gamma'_1, \Gamma'_2 \vdash e : \tau$. Let $\Gamma = \Gamma'_1, \Gamma'_2$. Then by Definition 8, we have $\vdash_R H_{\Sigma,D,g} : \Gamma$ (let Γ_i be $\emptyset$ in the definition). Thus, we get $\vdash_R \langle H_{\Sigma,D,g}, e\rangle : \tau$ as required.

Lemma 4 (progress). If $\vdash_R \langle H, e\rangle : \tau$, then either e is a value or there exist H' and e' such that $\langle H, e\rangle \longrightarrow \langle H', e'\rangle$.

Proof. Suppose $\vdash_R \langle H, e\rangle : \tau$, $\langle H, e\rangle \not\longrightarrow$, and e is not a value. Then, it must be one of the following cases:

- $e = E[d\langle\langle v\rangle\rangle]$ but v is neither a security variable σ nor an integer.
- $e = E[d\langle\langle \sigma\rangle\rangle]$ and $H(\sigma) = (n, p)$ but $p(d) = 0$ is undefined.
- $e = E[yv]$ and $H(y)$ is not a λ-abstraction.
- $e = E[yv]$ and $H(y) = \lambda^0 x.e'$.
- $e = E[\textbf{if } v \textbf{ then } e_1 \textbf{ else } e_2]$ but $val(H, v)$ is undefined.
- $e = E[v_1 \oplus v_2]$ but $val(H, v_1)$ or $val(H, v_2)$ is undefined.
- $e = E[\#_i(v)]$ but v is not of the form $\langle v_1, \ldots, v_n\rangle$ where $1 \leq i \leq n$.

By the assumption $\vdash_R \langle H, e\rangle$, none of the above cases cannot happen.

Lemma 5 (preservation). If $\vdash_R \langle H, e\rangle : \tau$ and $\langle H, e\rangle \longrightarrow \langle H', e'\rangle$, then $\vdash_R \langle H', e'\rangle : \tau$

Proof. See Section B.1.

Theorem 1 follows immeidately from the above lemmas.

Proof of S uppose $dom(\Sigma) = \{\sigma_1, \ldots, \sigma_k\}$ and $\vdash \langle \Sigma, D, e\rangle$ with $\langle H_{\Sigma,D,\{\sigma_1 \mapsto n_1, \ldots, \sigma_k \mapsto n_k\}}, e\rangle \longrightarrow^* \langle H, e'\rangle \not\longrightarrow$. By Lemmas 3 and 5, we have $\vdash_R \langle H, e'\rangle$. By Lemma 4, e' must be a value. □

B.1 Proof of Lemma 5

Lemma 6 (substitution). If $\Gamma_1, x : \tau' \vdash e : \tau \,\&\, \varphi$ and $\Gamma_2 \vdash v : \tau' \,\&\, \mathtt{t}$, then $\Gamma_1 + \Gamma_2 \vdash [v/x]e : \tau \,\&\, \varphi$.

Proof. This follows by straightforward induction on derivation of $\Gamma_1, x : \tau' \vdash e : \tau \,\&\, \varphi$. □

We define the summation $H_1 + H_2$ of heaps. $H_1 + H_2$ is defined only if whenever $x \in dom(H_1) \cap dom(H_2)$, $H_1(x)$ and $H_2(x)$ are identical except their uses or security levels. In that case, $H_1 + H_2$ is defined by:

$$
\begin{array}{l}
(H_1 + H_2)(v) = H_1(v) \text{if } v \in dom(H_1) \setminus dom(H_2) \\
(H_1 + H_2)(v) = H_2(v) \text{if } v \in dom(H_2) \setminus dom(H_1) \\
(H_1 + H_2)(\sigma) = (n, p_1 +\!\!+\, p_2) \text{if } H_1(\sigma) = (n, p_1) \text{ and } H_2(\sigma) = (n, p_2) \\
(H_1 + H_2)(f) = \lambda^{u_1 +\!\!+ u_2} x.e \text{if } H_1(f) = \lambda^{u_1} x.e \text{ and } H_2(f) = \lambda^{u_2} x.e
\end{array}
$$

Here, $+\!\!+$ is a restriction of $+$ such that $1 +\!\!+ 1$, $1 +\!\!+ \omega$, and $\omega +\!\!+ 1$ are undefined, so that if $u_1 +\!\!+ u_2 = \omega$ then $(u_1, u_2) \in \{(0, \omega), (\omega, 0), (\omega, \omega)\}$.

Lemma 7 (heap decomposition). *If $\vdash_R H : \Gamma_1 + \Gamma_2$, then there exist H_1 and H_2 such that $\vdash H_i : \Gamma_i$ ($i = 1, 2$) and $H_1 + H_2 = H$.*

Proof. This follows from the definition of $\vdash_R H : \Gamma$. □

Lemma 8. *Suppose that $H_1 + H_2$ and $\Gamma_1 + \Gamma_2$ are well-defined. If $\vdash_R H_1 : \Gamma_1$ and $\vdash_R H_2 : \Gamma_2$, then $\vdash_R H_1 + H_2 : \Gamma_1 + \Gamma_2$.*

Lemma 9. *If $\Gamma \vdash e : \tau \,\&\, \varphi$ and $\Gamma' \leq \Gamma$, then $\Gamma' \vdash e : \tau \,\&\, \varphi$.*

Proof. This follows by straightforward induction on the derivation of $\Gamma \vdash e : \tau \,\&\, \varphi$.

We now prove Lemma 5.

Proof of Lemma 5 Suppose $\vdash_R \langle H, e\rangle : \tau$ and $\langle H, e\rangle \longrightarrow \langle H', e'\rangle$. By the definition of $\vdash_R \langle H, e\rangle : \tau$, we have $\vdash_R H : \Gamma$ and $\Gamma \vdash e : \tau \,\&\, \varphi$ for some Γ. We show $\vdash_R \langle H', e\rangle$ by induction on the derivation of $\Gamma \vdash e : \tau \,\&\, \varphi$, with case analysis on the last rule used. Since the whole proof is basically the same as the soundness proof for other linear type systems, we show only the main cases below.

- Case T-App: In this case, $e = e_1 e_2$, $\Gamma_1 \vdash e_1 : \tau' \xrightarrow{\varphi_0}_1 \tau \,\&\, \varphi_1$, and $\Gamma_2 \vdash e_2 : \tau' \,\&\, \varphi_2$ with $\Gamma = \Gamma_1 + \Gamma_2$. By the assumption $\langle H, e\rangle \longrightarrow \langle H', e'\rangle$, there are three cases to consider.

- Case where $\langle H, e_1\rangle \longrightarrow \langle H', e_1'\rangle$ and $e' = e_1' e_2$.
  By Lemmas 7 and 9, there exist H_1 and H_2 such that:

  $$H_1 + H_2 = H \qquad\qquad \vdash_R H_i : \Gamma_i (i = 1, 2)$$

  By Lemma 4, $\langle H_1, e_1\rangle$ cannot get stuck, so that there exists H_1' such that $\langle H_1, e_1\rangle \longrightarrow \langle H_1', e_1'\rangle$ and $H' = H_1' + H_2$. By the induction hypothesis, we have $\vdash_R \langle H_1', e_1'\rangle$, so that there exists Γ_1' such that $\vdash_R H_1' : \Gamma_1'$ and $\Gamma_1' \vdash e_1' : \tau' \xrightarrow{\varphi_0}_1 \tau \,\&\, \varphi_1$. Let $\Gamma = \Gamma_1' + \Gamma_2$. Then, we have $\vdash_R H' : \Gamma'$ (by Lemma 8 and $\Gamma' \vdash e' : \tau \,\&\, \varphi$, from which $\vdash_R \langle H', e'\rangle : \tau$ follows.
- Case where $\langle H, e_2\rangle \longrightarrow \langle H', e_2'\rangle$ and $e' = e_1 e_2'$.
  Similar to the above case.
- Case where $e_1 = y$ and $H(y) = \lambda^u x.e_3$. In this case, e_2 is a value, $H' = H\{y \mapsto \lambda^{u-1} x.e_3\}$, and $e' = [e_2/x]e_3$.
  Let $H = \{y \mapsto \lambda^u x.e_3, f_1 \mapsto v_1, \ldots, f_k \mapsto v_k, \sigma_1 \mapsto (n_1, p_1), \ldots, \sigma_m \mapsto (n_m, p_m)\}$. Then, by $\vdash_R \langle H, ye_2\rangle$, we can assume without loss of generality:

  $\Gamma_0 \vdash \lambda^u x.e_3 : \tau' \xrightarrow{\varphi_0}_u \tau$
  $\Gamma_i' \vdash v_i : \tau_i'$ (for each $i = 1, \ldots, k$)
  $y : \tau' \xrightarrow{\varphi_0}_u \tau, f_1 : \tau_i', \ldots, f_k : \tau_k', \sigma_1 : int_{p_1}, \ldots, \sigma_m : int_{p_m} \leq \Gamma + \Gamma_0 + \Gamma_1' + \cdots + \Gamma_k'$
  $\Gamma = (y : \tau' \xrightarrow{\varphi_0}_1 \tau) + \Gamma_2$

  Since $u - 1$ is well-defined, it must be the case that $u \geq 1$. Therefore, the first condition impies that $\Gamma_0, x : \tau' \vdash e_3 : \tau \,\&\, \varphi_0$ (here, we use Lemma 9 when $u = \omega$). By Lemma 6, we get

  $$\Gamma_0 + \Gamma_2 \vdash [e_2/x]e_3 : \tau \,\&\, \varphi_0.$$

  Thus, the required result holds if we show $\vdash_R H' : \Gamma_0 + \Gamma_2$.
  If $u = \omega$, then Γ_0 is of the form $\omega \cdot \Gamma_0'$, so that $\Gamma_0 + \Gamma_0 = \Gamma_0$. Therefore,

  $$\begin{aligned} &y : \tau' \xrightarrow{\varphi_0}_u \tau, f_1 : \tau_i', \ldots, f_k : \tau_k', \sigma_1 : int_{p_1}, \ldots, \sigma_m : int_{p_m} \\ &\leq \Gamma + \Gamma_0 + \Gamma_1' + \cdots + \Gamma_k' \\ &= (\Gamma_0 + \Gamma) + \Gamma_0 + \Gamma_1' + \cdots + \Gamma_k' \\ &\leq (\Gamma_0 + \Gamma_2) + \Gamma_0 + \Gamma_1' + \cdots + \Gamma_k' \end{aligned}$$

  Thus, we have $\vdash_R H' (= H) : \Gamma_0 + \Gamma_2$ as required.
  If $u = 1$, then we have $0 \cdot \Gamma_0 \vdash \lambda^0 x.e_3 : \tau' \xrightarrow{\varphi_0}_0 \tau$. Therefore,

  $$\begin{aligned} &y : \tau' \xrightarrow{\varphi_0}_0 \tau, f_1 : \tau_i', \ldots, f_k : \tau_k', \sigma_1 : int_{p_1}, \ldots, \sigma_m : int_{p_m} \\ &\leq \Gamma_2 + \Gamma_0 + \Gamma_1' + \cdots + \Gamma_k' \\ &= (\Gamma_0 + \Gamma_2) + \Gamma_1' + \cdots + \Gamma_k' \end{aligned}$$

  Thus, we have $\vdash_R H' : \Gamma_0 + \Gamma_2$ as required.

– Case T-Dcl: In this case, $e = d\langle\langle e_1\rangle\rangle$, with $\Gamma(d) = int_{\mathbf{L}} \xrightarrow{\varphi_0}_\omega \tau$) and $\Gamma \vdash e_1 : int_{\{d \mapsto 1\}} \,\&\, \varphi_1$. By the assumption $\langle H, e\rangle \longrightarrow \langle H', e'\rangle$, there are three cases to consider.

- Case where $\langle H, e_1\rangle \longrightarrow \langle H', e_1'\rangle$ with $e' = d\langle\!\langle e_1'\rangle\!\rangle$.
  This follows easily from the induction hypothesis.
- Case where $e_1 = \sigma$. In this case, we have:

$$\begin{array}{ll} H(d) = \lambda^{\omega} x.e_d & e' = [n/x]e_d \\ H(\sigma) = (n, p) & H' = H\{\sigma \mapsto (n, p-d)\} \end{array}$$

  Let H be: $\{d \mapsto \lambda^{\omega} x.e_d, \sigma \mapsto (n,p), f_1 \mapsto v_1, \ldots, f_k \mapsto v_k, \sigma_1 \mapsto (n_1, p_1), \ldots, \sigma_m \mapsto (n_m, p_m)\}$. Then, by $\vdash_R \langle H, d\langle\!\langle\sigma\rangle\!\rangle\rangle$, we can assume without loss of generality:

$$\begin{array}{l} \emptyset \vdash \lambda^{\omega} x.e_d : int_{\mathbf{L}} \stackrel{\varphi_1}{\rightarrow}_{\omega} \tau \\ \Gamma_i' \vdash v_i : \tau_i' \text{ (for each } i = 1, \ldots, k) \\ \sigma : int_p, y : \tau' \stackrel{\varphi_0}{\rightarrow}_u \tau, f_1 : \tau_i', \ldots, f_k : \tau_k', \sigma_1 : int_{p_1}, \ldots, \sigma_m : int_{p_m} \leq \Gamma + \Gamma_1' + \cdots + \Gamma_k' \\ \Gamma = \sigma : int_{\{d \mapsto 1\}}, d : int_{\mathbf{L}} \stackrel{\varphi_1}{\rightarrow}_{\omega} \tau \end{array}$$

  By the first condition and Lemma 6, we get $\emptyset \vdash [n/x]e_d : \tau \,\&\, \varphi_1$. Thus, the required result follows if we show $\vdash_R H' : \emptyset$.
  Since

$$\begin{array}{l} \sigma : int_{p-d}, y : \tau' \stackrel{\varphi_0}{\rightarrow}_u \tau, f_1 : \tau_i', \ldots, f_k : \tau_k', \sigma_1 : int_{p_1}, \ldots, \sigma_m : int_{p_m} \\ \leq \sigma : int_{\{d \mapsto 0\}}, d : int_{\mathbf{L}} \stackrel{\varphi_1}{\rightarrow}_{\omega} \tau) + \Gamma_1' + \cdots + \Gamma_k' \\ \leq d : int_{\mathbf{L}} \stackrel{\varphi_1}{\rightarrow}_{\omega} \tau) + \Gamma_1' + \cdots + \Gamma_k' \end{array}$$

  we have $\vdash_R H' : \emptyset$ as required.
- Case where $e_1 = n$. Similar to the above case.

□

□

C Proof of Theorem 2

In this section, we assume that all the security levels except **L** and **H** have the same domain $\{d_1, \ldots, d_k\}$. $\{d_j \mapsto u_j\}$ is identified with $\{d_1 \mapsto 0, \ldots, d_{j-1} \mapsto 0, d_j \mapsto u_j, d_{j+1} \mapsto 0, \ldots . d_k \mapsto 0\}$. We first define the encoding of types and type environments by:

$$\begin{array}{l} [\![int_{\mathbf{L}}]\!]_{\Gamma} = int \\ [\![int_{\mathbf{H}}]\!]_{\Gamma} = \langle\,\rangle \\ [\![int_{\{d_1 \mapsto u_1, \ldots, d_k \mapsto u_k\}}]\!]_{\Gamma} = (\langle\,\rangle \rightarrow_{u_1} \tau_1) \times \cdots \times (\langle\,\rangle \rightarrow_{u_k} \tau_k) \\ \qquad \text{where } \Gamma(d_i) = int_{\mathbf{L}} \stackrel{\varphi}{\rightarrow}_{\omega} \tau_i \\ [\![\tau_1 \stackrel{\varphi}{\rightarrow}_u \tau_2]\!]_{\Gamma} = [\![\tau_1]\!]_{\Gamma} \stackrel{\varphi}{\rightarrow}_u [\![\tau_2]\!]_{\Gamma} \\ [\![x_1 : \tau_1, \ldots, x_n : \tau_n]\!] = x_1 : [\![\tau_1]\!]_{x_1:\tau_1, \ldots, x_n:\tau_n}, \ldots, x_n : [\![\tau_n]\!]_{x_1:\tau_1, \ldots, x_n:\tau_n} \end{array}$$

We say that a type environment Γ is *valid* if all the security levels in $\Gamma(d)$ are **L** for any declassification function variable d. We assume that type environments are always valid below.

The set of *extended values*, ranged over by V, is defined by:

$$V ::= f \mid n \mid \sigma \mid \langle V_1, \ldots, V_n \rangle \mid \lambda^u x.e$$

We define term transformation relation $\Gamma \vdash e : \tau \rightsquigarrow e'$, so that if $\Gamma \vdash e : \tau$ and $\Gamma \vdash e : \tau \rightsquigarrow e'$, then $[\![\Gamma]\!] \vdash e' : [\![\tau]\!]_\Gamma$. The transformation rules are given in Figure 5. In the figure, **let** $x = e_1$ **in** e_2 is an abbreviated form of $(\lambda^1 x.e_2)e_1$. We do not define transformation for a security variable σ since it is unnecessary in the proof.

We first prove properties of the above transformation.

Lemma 10. *If $\Gamma \vdash e : \tau \rightsquigarrow e'$ and $x \notin dom(\Gamma)$, then $\Gamma, x : \tau' \vdash e : \tau \rightsquigarrow e'$.*

Proof. Straightforward induction on the derivation of $\Gamma \vdash e : \tau \rightsquigarrow e'$. □

The following lemmas state that a well-typed term can be always transformed into a well-typed term.

Lemma 11. *If $\Gamma \vdash e : \tau$ and $\boldsymbol{SVar}(e) = \emptyset$, then there exists e' such that $\Gamma \vdash e : \tau \rightsquigarrow e'$.*

Proof. Straightforward induction on the derivation of $\Gamma \vdash e : \tau$. Note that each transformation rule is the same as the corresponding typing rule if the part "$\rightsquigarrow e'$" is ignored. □

In the above lemma, we can assume without loss of generality that Tr-SubV and Tr-SubE are not applied consecutively, because if consecutive applications of T-Sub in a type derivation can be always replaced by a single application of T-Sub. Thus, in the rest of this section, we assume that $\Gamma \vdash e : \tau \,\&\, \varphi \rightsquigarrow e'$ has been derived without applying Tr-SubV or Tr-SubE consecutively.

Lemma 12. *If $\Gamma \vdash e : \tau \rightsquigarrow e'$, then $\Gamma \vdash e : \tau$ and $[\![\Gamma]\!] \vdash e' : [\![\tau]\!]_\Gamma$.*

Proof. Suppose $\Gamma \vdash e : \tau \rightsquigarrow e'$. $\Gamma \vdash e : \tau$ follows immediately from the fact that each transformation rule coincides with a typing rule if the part "$\rightsquigarrow e'$" is ignored. $[\![\Gamma]\!] \vdash e' : [\![\tau]\!]_\Gamma$ also follows by straightforward induction on the derivation of $\Gamma \vdash e : \tau \rightsquigarrow e'$.

The following lemma states that a term of low-security type can be transformed into itself.

Lemma 13. *If $\emptyset \vdash e : \tau$ and if all the security levels in τ are* **L***, then $\emptyset \vdash e : \tau \rightsquigarrow e$.*

Proof. Let $toL(\Gamma)$ and $toL(\tau)$ be the type environment and type obtained from Γ and τ respectively, by replacing all the security levels with **L**. The lemma follows from the following more general lemma, which can be proved easily by induction on derivation of $\Gamma \vdash e : \tau$.

If $\Gamma \vdash e : \tau$ and if $dom(\Gamma) \cap \mathcal{N}_D = \emptyset$, then $toL(\Gamma) \vdash e : toL(\tau)$.

$$\frac{}{\Gamma, x : int_p \vdash x : int_p \,\&\, \mathtt{t} \rightsquigarrow x} \quad \text{(Tr-Var)}$$

$$\frac{}{\Gamma \vdash n : int_{\mathbf{L}} \,\&\, \mathtt{t} \rightsquigarrow n} \quad \text{(Tr-Const)}$$

$$\frac{\Gamma \vdash e : int_{\{d_j \mapsto 1\}} \,\&\, \varphi_1 \rightsquigarrow e'}{(d_j : int_{\mathbf{L}} \xrightarrow{\varphi_0}_{\omega} \tau) + \Gamma \vdash d_j\langle\!\langle e \rangle\!\rangle : \tau \,\&\, \varphi_0 \vee \varphi_1 \rightsquigarrow \#_j(e')\ \langle\,\rangle} \quad \text{(Tr-Dcl)}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \,\&\, \varphi \rightsquigarrow e'}{u \cdot \Gamma \vdash \lambda^u x.e : \tau_1 \xrightarrow{\varphi}_u \tau_2 \,\&\, \mathtt{t} \rightsquigarrow \lambda^u x.e'} \quad \text{(Tr-Fun)}$$

$$\frac{\Gamma, x : \tau_1 \xrightarrow{\mathtt{nt}}_{\omega} \tau_2, y : \tau_1 \vdash e : \tau_2 \,\&\, \varphi \rightsquigarrow e'}{\omega \cdot \Gamma \vdash \mathbf{fix}\ x(y) = e : \tau_1 \xrightarrow{\varphi}_{\omega} \tau_2 \,\&\, \mathtt{t} \rightsquigarrow \mathbf{fix}\ x(y) = e'} \quad \text{(Tr-Fix)}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \xrightarrow{\varphi_0}_1 \tau_2 \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma_2 \vdash e_2 : \tau_1 \,\&\, \varphi_2 \rightsquigarrow e_2'}{\Gamma_1 + \Gamma_2 \vdash e_1\ e_2 : \tau_2 \,\&\, \varphi_0 \vee \varphi_1 \vee \varphi_2 \rightsquigarrow e_1'\ e_2'} \quad \text{(Tr-App)}$$

$$\frac{\Gamma_1 \vdash e_1 : int_{\mathbf{L}} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma_2 \vdash e_2 : int_{\mathbf{L}} \,\&\, \varphi_2 \rightsquigarrow e_2'}{\Gamma_1 + \Gamma_2 \vdash e_1 \oplus e_2 : int_{\mathbf{L}} \,\&\, \varphi_1 \vee \varphi_2 \rightsquigarrow e_1' \oplus e_2'} \quad \text{(Tr-Op)}$$

$$\frac{\Gamma_1 \vdash e_1 : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma_2 \vdash e_2 : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_2' \qquad \lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}}{\Gamma_1 + \Gamma_2 \vdash e_1 \oplus e_2 : int_{\mathbf{H}} \,\&\, \varphi_1 \vee \varphi_2 \rightsquigarrow \mathbf{let}\ x_1 = e_1'\ \mathbf{in\ let}\ x_2 = e_2'\ \mathbf{in}\ \langle\,\rangle} \quad \text{(Tr-OpH1)}$$

$$\frac{\Gamma_1 \vdash V_1 : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma_2 \vdash e_2 : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_2' \qquad \lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}}{\Gamma_1 + \Gamma_2 \vdash V_1 \oplus e_2 : int_{\mathbf{H}} \,\&\, \varphi_1 \vee \varphi_2 \rightsquigarrow \mathbf{let}\ x = e_2'\ \mathbf{in}\ \langle\,\rangle} \quad \text{(Tr-OpH2)}$$

$$\frac{\Gamma_1 \vdash e_0 : int_{\mathbf{L}} \,\&\, \varphi_0 \rightsquigarrow e_0' \qquad \Gamma_2 \vdash e_1 : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma_2 \vdash e_2 : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_2'}{\Gamma_1 + \Gamma_2 \vdash \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : int_{p_1 \sqcup p_2} \,\&\, \varphi_0 \vee \varphi_1 \vee \varphi_2 \rightsquigarrow \mathbf{if}\ e_0'\ \mathbf{then}\ e_1'\ \mathbf{else}\ e_2'} \quad \text{(Tr-If)}$$

$$\frac{\begin{array}{c}\Gamma_1 \vdash e_0 : int_{p_0} \,\&\, \varphi \rightsquigarrow e_0' \qquad \Gamma_2 \vdash e_1 : int_{p_1} \,\&\, \mathtt{t} \rightsquigarrow e_1' \\ \Gamma_2 \vdash e_2 : int_{p_2} \,\&\, \mathtt{t} \rightsquigarrow e_2' \qquad \lceil p_0 \rceil = \mathbf{H}\end{array}}{\Gamma_1 + \Gamma_2 \vdash \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : int_{\mathbf{H}} \rightsquigarrow \mathbf{let}\ x = e_0'\ \mathbf{in}\ \langle\,\rangle} \quad \text{(Tr-IfH)}$$

$$\frac{\Gamma_i \vdash e_i : \tau_i \,\&\, \varphi_i \rightsquigarrow e_i' \text{ (for each } i \in \{1, \ldots, n\})}{\Gamma_1 + \cdots + \Gamma_n \vdash \langle e_1, \ldots, e_n \rangle : \langle \tau_1, \ldots, \tau_n \rangle \,\&\, \varphi_1 \vee \cdots \vee \varphi_n \rightsquigarrow \langle e_1', \ldots, e_n' \rangle} \quad \text{(Tr-Tuple)}$$

$$\frac{\Gamma \vdash V : \tau' \,\&\, \varphi' \rightsquigarrow e' \qquad \tau' \leq \tau \qquad \varphi' \leq \varphi}{\Gamma \vdash V : \tau \,\&\, \varphi \rightsquigarrow \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(e')} \quad \text{(Tr-SubV)}$$

$$\frac{\begin{array}{c}\Gamma \vdash e : \tau' \,\&\, \varphi' \rightsquigarrow e' \qquad \tau' \leq \tau \qquad \varphi' \leq \varphi \\ e \text{ is neither a value nor a variable}\end{array}}{\Gamma \vdash e : \tau \,\&\, \varphi \rightsquigarrow \mathbf{let}\ x = e'\ \mathbf{in\ coerce}_{\tau' \rightsquigarrow \tau}(x)} \quad \text{(Tr-SubE)}$$

$\mathbf{coerce}_{\tau \rightsquigarrow \tau'}(e)$ is defined by:

$$\mathbf{coerce}_{int_{\mathbf{L}} \rightsquigarrow int_{\{d_1 \mapsto u_1, \ldots, d_k \mapsto u_k\}}}(e) = \langle \lambda^{u_1} x.(d_1\langle\!\langle e \rangle\!\rangle), \ldots, \lambda^{u_k} x.(d_k\langle\!\langle e \rangle\!\rangle) \rangle$$

$$\mathbf{coerce}_{int_{\{d_1 \mapsto u_1, \ldots, d_k \mapsto u_k\}} \rightsquigarrow int_{\{d_1 \mapsto u_1', \ldots, d_k \mapsto u_k'\}}}(e) = e$$

$$\mathbf{coerce}_{int_p \rightsquigarrow int_{\mathbf{H}}}(e) = \langle\,\rangle$$

$$\mathbf{coerce}_{\tau_1 \to_u \tau_2 \rightsquigarrow \tau_1' \to_{u'} \tau_2'}(e) = \lambda^{u'} x.\mathbf{let}\ y = e(\mathbf{coerce}_{\tau_1' \rightsquigarrow \tau_1}(x))\ \mathbf{in\ coerce}_{\tau_2 \rightsquigarrow \tau_2'}(y)$$

Fig. 5. Transformation Rules

□

The following lemma states that the transformation preserves the semantics of terms. We shall prove the lemma later, in Subsection C.1.

Lemma 14. *Let D be a declassification environment. If $\vdash D : \Gamma$ and $\Gamma \vdash e : int_{\mathbf{L}} \,\&\, \varphi \rightsquigarrow e'$, then $\langle D, e\rangle \Downarrow 0$ if and only if $\langle D, e'\rangle \Downarrow 0$.*

We are now ready to prove Theorem 2.

Proof of Theorem 2 Suppose that $\vdash \langle \Sigma, D, e\rangle : \tau$ and all the security levels in τ are $\mathbf{L}$. Suppose also that $\Sigma = \{\sigma_1 \mapsto \{d_1 \mapsto u_{11}, \ldots, d_k \mapsto u_{1k}\}, \ldots, \sigma_m \mapsto \{d_1 \mapsto u_{m1}, \ldots, d_k \mapsto u_{mk}\}\}$. Then there exist Γ_1 and Γ_2 such that:

$$\vdash \Sigma : \Gamma_1 \qquad \vdash D : \Gamma_2 \qquad \Gamma_1, \Gamma_2 \vdash e : \tau$$

Let $\Gamma_1' = x_1 : \Gamma_1(\sigma_1), \ldots, x_m : \Gamma_1(\sigma_m)$ and $e_0 = [x_1/\sigma_1, \ldots, x_m/\sigma_m]e$. Then, we have $\Gamma_1', \Gamma_2 \vdash e_0 : \tau$.

By Lemma 11, there exists e_0' such that $\Gamma_1', \Gamma_2 \vdash e_0 : \tau \rightsquigarrow e_0'$. Let e' be $D(\lambda^1 x_1. \cdots \lambda^1 x_m. e_0')$. We shall show that e' satisfies the condition in Definition 7.

By the transformation rules, we have:

$$\begin{aligned}
&\Gamma_2 \vdash (\lambda^1 x_1. \cdots \lambda^1 x_m. e_0) n_1 \cdots n_m : \tau \\
&\quad \rightsquigarrow (\lambda^1 x_1. \cdots \lambda^1 x_m. e_0') \\
&\qquad \langle \lambda^{u_{11}} x. (d_1 \langle\!\langle n_1 \rangle\!\rangle), \ldots, \lambda^{u_{1k}} x. (d_k \langle\!\langle n_1 \rangle\!\rangle)\rangle \cdots \langle \lambda^{u_{m1}} x. (d_1 \langle\!\langle n_m \rangle\!\rangle), \ldots, \lambda^{u_{mk}} x. (d_k \langle\!\langle n_m \rangle\!\rangle)\rangle
\end{aligned}$$

Let e_1 be any term such that $\emptyset \vdash e_1 : \tau \xrightarrow{\mathtt{nt}}_{\omega} int_{\mathbf{L}}$. By Lemma 13 and the assumption that all the security levels in τ are $\mathbf{L}$, we have $\emptyset \vdash e_1 : \tau \xrightarrow{\mathtt{nt}}_{\omega} int_{\mathbf{L}} \rightsquigarrow e_1$. By using Tr-App, we obtain:

$$\begin{aligned}
&\Gamma_2 \vdash e_1((\lambda^1 x_1. \cdots \lambda^1 x_m. e_0) n_1 \cdots n_m) : int_{\mathbf{L}} \\
&\quad \rightsquigarrow e_1((\lambda^1 x_1. \cdots \lambda^1 x_m. e_0') \\
&\qquad \langle \lambda^{u_{11}} x. (d_1 \langle\!\langle n_1 \rangle\!\rangle), \ldots, \lambda^{u_{1k}} x. (d_k \langle\!\langle n_1 \rangle\!\rangle)\rangle \cdots \langle \lambda^{u_{m1}} x. (d_1 \langle\!\langle n_m \rangle\!\rangle), \ldots, \lambda^{u_{mk}} x. (d_k \langle\!\langle n_m \rangle\!\rangle)\rangle)
\end{aligned}$$

Thus, we obtain the following equivalence.

$$\begin{aligned}
&(\emptyset, e_1([n_1/\sigma_1, \ldots, n_m/\sigma_m]D(e))) \Downarrow 0 \\
\iff &(\emptyset, e_1((\lambda^1 x_1. \cdots \lambda^1 x_m. D(e_0)) n_1 \cdots n_m)) \Downarrow 0 \\
\iff &(D, e_1((\lambda^1 x_1. \cdots \lambda^1 x_m. e_0) n_1 \cdots n_m)) \Downarrow 0 \\
\iff &(D, e_1((\lambda^1 x_1. \cdots \lambda^1 x_m. e_0') \\
&\quad \langle \lambda^{u_{11}} x. (d_1 \langle\!\langle n_1 \rangle\!\rangle), \ldots, \lambda^{u_{1k}} x. (d_k \langle\!\langle n_1 \rangle\!\rangle)\rangle \cdots \langle \lambda^{u_{m1}} x. (d_1 \langle\!\langle n_m \rangle\!\rangle), \ldots, \lambda^{u_{mk}} x. (d_k \langle\!\langle n_m \rangle\!\rangle)\rangle))) \Downarrow 0 \\
\iff &(\emptyset, e_1(e' \\
&\quad \langle \lambda^{u_{11}} x. (D(d_1) n_1), \ldots, \lambda^{u_{1k}} x. (D(d_k) n_1)\rangle \cdots \langle \lambda^{u_{m1}} x. (D(d_1) n_m), \ldots, \lambda^{u_{mk}} x. (D(d_k) n_m)\rangle)) \Downarrow 0
\end{aligned}$$

Here, the third equivalence follows from Lemma 14. Therefore, we have:

$$\begin{aligned}
[n_1/\sigma_1, \ldots, n_m/\sigma_m]D(e) \approx_{\tau, \mathtt{nt}} e' \; &\langle \lambda^{u_{11}} x. (D(d_1) n_1), \ldots, \lambda^{u_{1k}} x. (D(d_k) n_1)\rangle \\
&\cdots \\
&\langle \lambda^{u_{m1}} x. (D(d_1) n_m), \ldots, \lambda^{u_{mk}} x. (D(d_k) n_m)\rangle
\end{aligned}$$

as required. □

C.1 Proof of Lemma 14

To prove the lemma, we introduce a substitution-based reduction relation $e \longrightarrow_D e'$, which is insensitive to the linearity information.

$$E[d\langle\!\langle n \rangle\!\rangle] \longrightarrow_D E[D(d)n] \quad \text{(R-Declassify)}$$
$$E[(\lambda^u x.e)V] \longrightarrow_D E[[V/x]e] \quad \text{(R-App)}$$
$$\frac{n \neq 0}{E[\textbf{if } n \textbf{ then } e_1 \textbf{ else } e_2] \longrightarrow_D E[e_1]} \quad \text{(R-IfT)}$$
$$E[\textbf{if } 0 \textbf{ then } e_1 \textbf{ else } e_2] \longrightarrow_D E[e_2] \quad \text{(R-IfF)}$$
$$E[n_1 \oplus n_2] \longrightarrow_D E[n_1 \underline{\oplus} n_2] \quad \text{(R-Op)}$$
$$E[\textbf{fix } x(y) = e] \longrightarrow_D E[\lambda^\omega y.[\textbf{fix } x(y) = e/x]e] \quad \text{(R-Fix)}$$
$$E[\#_i(V_1, \ldots, V_m)] \longrightarrow_D E[V_i] \quad \text{(R-Proj)}$$

Evaluation contexts:

$$E ::= [\,] \mid [\,]e \mid V[\,] \mid d\langle\!\langle [\,] \rangle\!\rangle \mid \textbf{if } [\,] \textbf{ then } e_1 \textbf{ else } e_2 \mid [\,] \oplus e \mid v \oplus [\,] \\ \mid \langle V_1, \ldots, V_{k-1}, [\,], e_{k+1}, \ldots, e_n \rangle \mid \#_i([\,])$$

Fig. 6. Linearity-insensitive operational semantics

We also introduce a "linearity-insensitive" version of the transformation relation $\Gamma \vdash_\omega e : \tau \,\&\, \varphi \rightsquigarrow e'$. The rules for $\Gamma \vdash_\omega e : \tau \,\&\, \varphi \rightsquigarrow e'$ are obtained as a restriction of the rules for $\Gamma \vdash e : \tau \,\&\, \varphi \rightsquigarrow e'$, where all the uses occuring in expressions and types must be ω. Similarly, we also write $\vdash_\omega D : \Gamma$ for a linearity-insensive version of type judgment, where all the uses in its derivation must be ω.

Lemma 15 (substitution lemma for linearity-insensitive transformation). *If $\Gamma, x : \tau' \vdash_\omega e : \tau \,\&\, \varphi \rightsquigarrow e'$ and $\Gamma \vdash_\omega V : \tau' \,\&\, \mathtt{t} \rightsquigarrow V'$, then $\Gamma \vdash_\omega [V/x]e : \tau \,\&\, \varphi \rightsquigarrow [V'/x]e'$*

Proof. This follows by straightforward induction on the derivation of $\Gamma, x : \tau' \vdash_\omega e : \tau \,\&\, \varphi \rightsquigarrow e'$.

The following lemma states that $\longrightarrow_D^*$ is preserved by the linearity-insensitive transformation relation.

Lemma 16. *Let D be a declassification environment. If $\vdash_\omega D : \Gamma$ and $\Gamma \vdash_\omega e_0 : \tau \,\&\, \varphi \rightsquigarrow e_1$, then the following conditions hold.*

1. *If $e_0 \longrightarrow_D e_0''$, then there exist e_0' and e_1' such that $e_0'' \longrightarrow_D^* e_0'$ and $e_1 \longrightarrow_D^* e_1'$ with $\Gamma \vdash_\omega e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$ for some e_1'.*
2. *If $e_1 \longrightarrow_D e_1''$, then there exist e_0' and e_1' such that $e_1'' \longrightarrow_D^* e_1'$ and $e_0 \longrightarrow_D^* e_0'$ with $\Gamma \vdash_\omega e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$.*

$$\frac{}{\Gamma, x : int_p \vdash_\omega x : int_p \,\&\, \mathtt{t} \rightsquigarrow x} \quad (\textsc{TrO-Var})$$

$$\frac{}{\Gamma \vdash_\omega n : int_{\mathbf{L}} \,\&\, \mathtt{t} \rightsquigarrow n} \quad (\textsc{TrO-Const})$$

$$\frac{\Gamma \vdash_\omega e : int_{\{d_j \mapsto \omega\}} \,\&\, \varphi_1 \rightsquigarrow e' \qquad \Gamma(d_j) = int_{\mathbf{L}} \xrightarrow{\varphi_0}_\omega \tau}{\Gamma \vdash_\omega d_j\langle\!\langle e \rangle\!\rangle : \tau \,\&\, \varphi_0 \vee \varphi_1 \rightsquigarrow \#_j(e')\ \langle\,\rangle} \quad (\textsc{TrO-Dcl})$$

$$\frac{\Gamma, x : \tau_1 \vdash_\omega e : \tau_2 \,\&\, \varphi \rightsquigarrow e'}{\Gamma \vdash_\omega \lambda^\omega x.e : \tau_1 \xrightarrow{\varphi}_\omega \tau_2 \,\&\, \mathtt{t} \rightsquigarrow \lambda^\omega x.e'} \quad (\textsc{TrO-Fun})$$

$$\frac{\Gamma, x : \tau_1 \xrightarrow{\mathtt{nt}}_\omega \tau_2, y : \tau_1 \vdash_\omega e : \tau_2 \,\&\, \varphi \rightsquigarrow e'}{\Gamma \vdash_\omega \mathbf{fix}\ x(y) = e : \tau_1 \xrightarrow{\varphi}_\omega \tau_2 \,\&\, \mathtt{t} \rightsquigarrow \mathbf{fix}\ x(y) = e'} \quad (\textsc{TrO-Fix})$$

$$\frac{\Gamma \vdash_\omega e_1 : \tau_1 \xrightarrow{\varphi_0}_\omega \tau_2 \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma \vdash_\omega e_2 : \tau_1 \,\&\, \varphi_2 \rightsquigarrow e_2'}{\Gamma \vdash_\omega e_1\ e_2 : \tau_2 \,\&\, \varphi_0 \vee \varphi_1 \vee \varphi_2 \rightsquigarrow e_1'\ e_2'} \quad (\textsc{TrO-App})$$

$$\frac{\Gamma \vdash_\omega e_1 : int_{\mathbf{L}} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma \vdash_\omega e_2 : int_{\mathbf{L}} \,\&\, \varphi_2 \rightsquigarrow e_2'}{\Gamma \vdash_\omega e_1 \oplus e_2 : int_{\mathbf{L}} \,\&\, \varphi_1 \vee \varphi_2 \rightsquigarrow e_1' \oplus e_2'} \quad (\textsc{TrO-Op})$$

$$\frac{\Gamma \vdash_\omega e_1 : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma \vdash_\omega e_2 : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_2' \qquad \lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H} \qquad e_1 \text{ is not a value or a variable}}{\Gamma \vdash_\omega e_1 \oplus e_2 : int_{\mathbf{H}} \,\&\, \varphi_1 \vee \varphi_2 \rightsquigarrow \mathbf{let}\ x_1 = e_1'\ \mathbf{in}\ \mathbf{let}\ x_2 = e_2'\ \mathbf{in}\ \langle\,\rangle} \quad (\textsc{TrO-OpH1})$$

$$\frac{\Gamma \vdash_\omega V_1 : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma \vdash_\omega e_2 : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_2' \qquad \lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}}{\Gamma \vdash_\omega V_1 \oplus e_2 : int_{\mathbf{H}} \,\&\, \varphi_1 \vee \varphi_2 \rightsquigarrow \mathbf{let}\ x = e_2'\ \mathbf{in}\ \langle\,\rangle} \quad (\textsc{TrO-OpH2})$$

$$\frac{\Gamma \vdash_\omega e_0 : int_{\mathbf{L}} \,\&\, \varphi_0 \rightsquigarrow e_0' \qquad \Gamma \vdash_\omega e_1 : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_1' \qquad \Gamma \vdash_\omega e_2 : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_2'}{\Gamma \vdash_\omega \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : int_{p_1 \sqcup p_2} \,\&\, \varphi_0 \vee \varphi_1 \vee \varphi_2 \rightsquigarrow \mathbf{if}\ e_0'\ \mathbf{then}\ e_1'\ \mathbf{else}\ e_2'} \quad (\textsc{TrO-If})$$

$$\frac{\Gamma \vdash_\omega e_0 : int_{p_0} \,\&\, \varphi \rightsquigarrow e_0' \qquad \Gamma \vdash_\omega e_1 : int_{p_1} \,\&\, \mathtt{t} \rightsquigarrow e_1' \qquad \Gamma \vdash_\omega e_2 : int_{p_2} \,\&\, \mathtt{t} \rightsquigarrow e_2' \qquad \lceil p_0 \rceil = \mathbf{H}}{\Gamma \vdash_\omega \mathbf{if}\ e_0\ \mathbf{then}\ e_1\ \mathbf{else}\ e_2 : int_{\mathbf{H}} \,\&\, \varphi \rightsquigarrow \mathbf{let}\ x = e_0'\ \mathbf{in}\ \langle\,\rangle} \quad (\textsc{TrO-IfH})$$

$$\frac{\Gamma \vdash_\omega e_i : \tau_i \,\&\, \varphi_i \rightsquigarrow e_i' \text{ (for each } i \in \{1, \ldots, n\})}{\Gamma \vdash_\omega \langle e_1, \ldots, e_n \rangle : \langle \tau_1, \ldots, \tau_n \rangle \,\&\, \varphi_1 \vee \cdots \vee \varphi_n \rightsquigarrow \langle e_1', \ldots, e_n' \rangle} \quad (\textsc{TrO-Tuple})$$

$$\frac{\Gamma \vdash_\omega V : \tau' \,\&\, \varphi' \rightsquigarrow e' \qquad \tau' \leq \tau \qquad \varphi' \leq \varphi}{\Gamma \vdash_\omega V : \tau \,\&\, \varphi \rightsquigarrow \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(e')} \quad (\textsc{TrO-SubV})$$

$$\frac{\Gamma \vdash_\omega e : \tau' \,\&\, \varphi' \rightsquigarrow e' \qquad \tau' \leq \tau \qquad \varphi' \leq \varphi \qquad e \text{ is not a value or a variable}}{\Gamma \vdash_\omega e : \tau \,\&\, \varphi \rightsquigarrow \mathbf{let}\ x = e'\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(x)} \quad (\textsc{TrO-SubE})$$

Fig. 7. Linearity-Insensitive Transformation Rules

Proof. In the proof below, we use the property that if $\vdash_\omega D : \Gamma$ and $\Gamma \vdash e : int_p \,\&\, \mathtt{t}$, then $e \longrightarrow_D^* n$. The proof of this property is omited.

1. The proof proceeds by induction on the derivation of $\Gamma \vdash_\omega e_0 : \tau \,\&\, \varphi \rightsquigarrow e_1$, with case analysis on the last rule used.
   - Case TrO-Var: This case cannot happen since Γ can contain only declassification variables.
   - Case TrO-Const: This case cannot happen, since e_0 must be n, which cannot be reduced.
   - Case TrO-Dcl: In this case, $e_0 = d_j\langle\!\langle e_{01} \rangle\!\rangle$ and $e_1 = \#_j(e_{11})\langle\,\rangle$ with $\Gamma \vdash_\omega e_{01} : int_{\{d_j \mapsto \omega\}} \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma(d_j) = int_{\mathbf{L}} \stackrel{\varphi_0}{\rightarrow}_\omega \tau$. By the assumption $e_0 \longrightarrow_D e_0''$, there are two cases to consider.
     - Case where $e_0'' = d_j\langle\!\langle e_{01}'' \rangle\!\rangle$ and $e_{01} \longrightarrow_D e_{01}''$. By the induction hypothesis, we get $e_{01}'' \longrightarrow_D^* e_{01}'$ and $e_{11} \longrightarrow_D^* e_{11}'$ with $\Gamma \vdash_\omega e_{01}' : int_{\{d_j \mapsto \omega\}} \,\&\, \varphi_1 \rightsquigarrow e_{11}'$. Thus, the required result holds for $e_0' = d_j\langle\!\langle e_{01}' \rangle\!\rangle$ and $e_1' = \#_j(e_{11}')\langle\,\rangle$.
     - Case where $e_{01} = n$ and $e_0'' = [n/y]e_{d_j}$, with $D(d_j) = \lambda^\omega y.e_{d_j}$. In this case, $\Gamma \vdash_\omega e_{01} : int_{\{d_j \mapsto \omega\}} \,\&\, \varphi_1 \rightsquigarrow e_{11}$ must have been derived by using TrO-SubV, so that e_{11} is of the form $\langle \lambda^\omega x.d_1\langle\!\langle n \rangle\!\rangle, \ldots, \lambda^\omega x.d_k\langle\!\langle n \rangle\!\rangle \rangle$. Let $e_0' = e_1' = e_0''(= [n/y]e_{d_j})$. Then, we have $e_1 \longrightarrow_D (\lambda^\omega x.d_j\langle\!\langle n \rangle\!\rangle)\langle\,\rangle \longrightarrow_D d_j\langle\!\langle n \rangle\!\rangle \longrightarrow_D e_1'$. Moreover, since e_0' is a closed expression, by using Lemma 13, we get $\emptyset \vdash_\omega e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$. Thus, we have $\Gamma \vdash_\omega e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$ as required.
   - Cases TrO-Fun: This case cannot happen, since e_0 must be a λ-abstraction, which contradicts with $e_0 \longrightarrow_D e_0''$.
   - Case TrO-Fix: In this case, $e_0 = \mathbf{fix}\ x(y) = e_{01}$ and $e_1 = \mathbf{fix}\ x(y) = e_{11}$ with $\Gamma, x : \tau_1 \stackrel{\mathtt{nt}}{\rightarrow}_\omega \tau_2, y : \tau_1 \vdash_\omega e_{01} : \tau_2 \,\&\, \varphi \rightsquigarrow e_{11}$. e_0'' must be $\lambda^\omega y.[\mathbf{fix}\ x(y) = e_{01}/x]e_{01}$. The required result holds for $e_0' = e_0''$ and $e_1' = \lambda^\omega y.[\mathbf{fix}\ x(y) = e_{11}/x]e_{11}$.
   - Case TrO-App: In this case $e_0 = e_{01}\ e_{02}$ and $e_1 = e_{11}\ e_{12}$ with $\Gamma \vdash_\omega e_{01} : \tau_1 \stackrel{\varphi_0}{\rightarrow}_\omega \tau_2 \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : \tau_1 \,\&\, \varphi_2 \rightsquigarrow e_{12}$.
     By the assumption $e_0 \longrightarrow_D e_0''$, there are three cases to consider.
     - Case where $e_0'' = e_{01}''e_{02}$ and $e_{01} \longrightarrow_D e_{01}'$. By the induction hypothesis, there exist e_{01}' and e_{11}' such that $e_{01}'' \longrightarrow_D^* e_{01}'$ and $e_{11} \longrightarrow_D^* e_{11}'$ with $\Gamma \vdash_\omega e_{01}' : \tau_1 \stackrel{\varphi_0}{\rightarrow}_\omega \tau_2 \,\&\, \varphi_2 \rightsquigarrow e_{11}'$. The required result holds for $e_0' = e_{01}'e_{02}$ and $e_1' = e_{11}'e_{12}$.
     - Case where $e_0'' = e_{01}e_{02}''$ and $e_{02} \longrightarrow_D e_{02}''$. Similar to the above case.
     - Case where $e_{01} = \lambda^\omega x.e_{03}$ and $e_0'' = [e_{02}/x]e_{03}$. In this case, e_{02} must be an extended value. If $\Gamma \vdash_\omega e_{01} : \tau_1 \stackrel{\varphi_0}{\rightarrow}_\omega \tau_2 \,\&\, \varphi_2 \rightsquigarrow e_{11}$ is derived by using TrO-Fun, then we have $\Gamma, x : \tau_1 \vdash_\omega e_{03} : \tau_2 \,\&\, \varphi_0 \rightsquigarrow e_{13}$ with $e_{11} = \lambda^\omega x.e_{13}$. Let $e_0' = e_0''$ and$e_1' = [e_{02}'/x]e_{13}$. Then, we have $e_0'' \longrightarrow_D^* e_0'$ and $e_1 \longrightarrow_D^* e_1'$. Moreover, by Lemma 15, we get $\Gamma \vdash_\omega e_1 : \tau_2 \,\&\, \varphi \rightsquigarrow e_1'$ as required.

If $\Gamma \vdash_\omega e_{01} : \tau_1 \xrightarrow{\varphi_0}_\omega \tau_2 \,\&\, \varphi_1 \rightsquigarrow e_{11}$ is derived by using TrO-SubV, then we have:

$\Gamma, x : \tau_1' \vdash_\omega e_{03} : \tau_2' \,\&\, \varphi_0' \rightsquigarrow e_{13}$
$e_{11} = \lambda^\omega x.\mathbf{let}\ y = (\lambda^\omega x.e_{13})\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(x)\ \mathbf{in}\ \mathbf{coerce}_{\tau_2' \rightsquigarrow \tau_2}(y)$
$\Gamma \vdash_\omega e_{02} : \tau_1 \,\&\, \varphi_2 \rightsquigarrow \mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})$
e_{02} and e_{12} are extended values

By applying Lemma 15 to the first and third conditions, we get:

$$\Gamma \vdash_\omega e_1' : \tau_2' \,\&\, \varphi_0' \rightsquigarrow [\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})/x]e_{13}.$$

The required result holds for $e_0' = e_0''$ and
$e_1' = \mathbf{let}\ y = [\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})/x]e_{13}\ \mathbf{in}\ \mathbf{coerce}_{\tau_2' \rightsquigarrow \tau_2}(y)$.

– Case TrO-Op: In this case, $e_0 = e_{01} \oplus e_{02}$ and $e_1 = e_{11} \oplus e_{12}$ with $\Gamma \vdash_\omega e_{01} : int_{\mathbf{L}} \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : int_{\mathbf{L}} \,\&\, \varphi_2 \rightsquigarrow e_{12}$. By the assumption $e_0 \longrightarrow_D e_0''$, there are three cases to consider.
  - Cases where $e_0'' = e_{01}'' \oplus e_{02}$ with $e_{01} \longrightarrow_D e_{01}''$; or $e_0'' = e_{01} \oplus e_{02}''$ with $e_{02} \longrightarrow_D e_{02}''$. These cases follow immediately from the induction hypothesis.
  - Case where $e_{01} = n_1$ and $e_{02} = n_2$ with $e_0'' = n_1 \underline{\oplus} n_2$. By the transformation rules, e_1 must be $n_1 \oplus n_2$. So, the required result holds for $e_0' = e_0''$ and $e_1' = n_1 \underline{\oplus} n_2$.

– Case TrO-OpH1: In this case, $e_0 = e_{01} \oplus e_{02}$ and $e_1 = \mathbf{let}\ x_1 = e_{11}\ \mathbf{in}\ \mathbf{let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\,\rangle$ with $\Gamma \vdash_\omega e_{01} : int_{p_1} \,\&\, \varphi_2 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_{12}$. Moreover, $\lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}$ and e_2 is not a value. Since e_2 is not a value, it must be the case that $e_0'' = e_{01}'' \oplus e_{02}$ with $e_{01} \longrightarrow_D e_{01}''$. By the induction hypothesis, there exist e_{01}' and e_{11}' such that $e_{01}'' \longrightarrow_D^* e_{01}'$ and $e_{11} \longrightarrow_D^* e_{11}'$ with $\Gamma \vdash_\omega e_{01}' : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_{11}'$. Let e_0' be $e_{01}' \oplus e_{02}$. Let us define e_1' by:

$$e_1' = \begin{cases} \mathbf{let}\ x_1 = e_{11}'\ \mathbf{in}\ \mathbf{let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\,\rangle & \text{if } e_{11} \text{ is not an extended value} \\ \mathbf{let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\,\rangle & \text{otherwise} \end{cases}$$

Then, we have $e_1 \longrightarrow_D^* e_1'$ and $\Gamma \vdash_\omega e_0' : int_{\mathbf{H}} \,\&\, \varphi \rightsquigarrow e_1'$ as required.

– Case TrO-OpH2: In this case, $e_0 = n_{01} \oplus e_{02}$ and $e_1 = \mathbf{let}\ x = e_{12}\ \mathbf{in}\ \langle\,\rangle$ with $\Gamma \vdash_\omega n_{01} : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow V_{11}'$ and $\Gamma \vdash_\omega e_{02} : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_{12}$. Moreover, $\lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}$. By the assumption $e_0 \longrightarrow_D e_0''$, there are two cases to consider.
  - Case where $e_0'' = n_{01} \oplus e_{02}''$ with and $e_{02} \longrightarrow_D e_{02}''$. By the induction hypothesis, there exist e_{02}' and e_{12}' such that $e_{02}'' \longrightarrow_D^* e_{02}'$ and $e_{12} \longrightarrow_D^* e_{12}'$ with $\Gamma \vdash_\omega e_{02}' : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_{12}'$. The required result holds for $e_0' = n_{01} \oplus e_{02}'$ and $e_1' = \mathbf{let}\ x = e_{12}'\ \mathbf{in}\ \langle\,\rangle$.
  - Case where $e_{02} = n_{02}$ and $e_0'' = n_{01} \underline{\oplus} n_{02}$. The required result holds for $e_0' = e_0''$ and $e_1' = \langle\,\rangle$.

– Case TrO-If: In this case, $e_0 = \mathbf{if}\ e_{00}\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1 = \mathbf{if}\ e_{10}\ \mathbf{then}\ e_{11}\ \mathbf{else}\ e_{12}$ with $\Gamma \vdash_\omega e_{00} : int_{\mathbf{L}} \,\&\, \varphi_0 \rightsquigarrow e_{10}$, $\Gamma \vdash_\omega e_{01} : int_{p_1} \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : int_{p_2} \,\&\, \varphi_2 \rightsquigarrow e_{12}$. By the assumption $e_0 \longrightarrow_D e_0''$, there are three cases to consider.

- Case where $e_0'' = \mathbf{if}\ e_{00}''\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_{00} \longrightarrow_D e_{00}''$. By the induction hypothesis, there exist e_{00}' and e_{10}' such that $e_{00}'' \longrightarrow_D^* e_{00}'$ and $e_{10} \longrightarrow_D^* e_{10}'$ with $\Gamma \vdash_\omega e_{00}' : int_{\mathbf{L}} \,\&\, \varphi_0 \rightsquigarrow e_{10}'$. The required result holds for $e_0' = \mathbf{if}\ e_{00}'\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1' = \mathbf{if}\ e_{10}'\ \mathbf{then}\ e_{11}\ \mathbf{else}\ e_{12}$.
- Case where $e_{00} = n (\neq 0)$ and $e_0'' = e_{01}$. By the transformation rule, e_{10} must be n. Thus, the required result holds for $e_0' = e_0''$ and $e_1' = e_{11}$.
- Case where $e_{00} = 0$ and $e_0' = e_{02}$. Similar to the above case.

– Case TrO-IfH: In this case, $e_0 = \mathbf{if}\ e_{00}\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1 = \mathbf{let}\ x = e_{10}\ \mathbf{in}\ \langle\rangle$ with $\Gamma \vdash_\omega e_{00} : int_{p_0} \,\&\, \varphi_2 \rightsquigarrow e_{10}$ and $\lceil p \rceil = \mathbf{H}$. By the assumption $e_0 \longrightarrow_D e_0''$, there are three cases to consider.
- Case where $e_0'' = \mathbf{if}\ e_{00}''\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_{00} \longrightarrow_D e_{00}''$.
  By the induction hypothesis, there exist e_{00}' and e_{10}' such that $e_{00}'' \longrightarrow_D^* e_{00}'$ and $e_{10} \longrightarrow_D^* e_{10}'$ with $\Gamma \vdash_\omega e_{00}' : int_{p_0} \,\&\, \varphi_0 \rightsquigarrow e_{10}'$. The required result holds for $e_0' = \mathbf{if}\ e_{00}'\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1' = \mathbf{let}\ x = e_{10}'\ \mathbf{in}\ \langle\rangle$.
- Case where $e_{00} = n (\neq 0)$ and $e_0'' = e_{01}$.
  Since $\Gamma \vdash_\omega e_{01} : int_{p_1} \,\&\, \mathtt{t}$, it must be the case that $e_{01} \longrightarrow_D^* n_{01}$ for some n_{01}. Let $e_0' = n_{01}$ and $e_1' = \langle\rangle$. Then, we have $\Gamma \vdash_\omega e_0' : int_{\mathbf{H}} \,\&\, \mathtt{t} \rightsquigarrow e_1'$ and $e_1 \longrightarrow_D^* e_1'$ as required.
- Case where $e_{00} = 0$ and $e_0'' = e_{02}$.
  Similar to the above case.

– Case TrO-Tuple: This case follows immediately from the induction hypothesis.

– Case TrO-SubV: This case cannot happen, since e_0 must be a value, which contradicts with $e_0 \longrightarrow_D e_0'$.

– Case TrO-SubE: In this case, $e_1 = \mathbf{let}\ x = e_{11}\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(x)$ and $\Gamma \vdash_\omega e_0 : \tau' \,\&\, \varphi' \rightsquigarrow e_{11}$, By the induction hypothesis, there exists e_{11}' such that $e_{11} \longrightarrow_D^* e_{11}'$ and $\Gamma \vdash_\omega e_0' \tau' \,\&\, \varphi' \rightsquigarrow e_{11}'$. Let e_1' be $\mathbf{coerce}_{\tau' \rightsquigarrow \tau}(e_{11}')$ if e_0' is a value, and be $\mathbf{let}\ x = e_{11}'\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(x)$ otherwise. Let e_0' be e_0''. Then, we get $e_0'' \longrightarrow_D^* e_0'$ and $e_1 \longrightarrow_D^* e_1'$ with $\Gamma \vdash_\omega e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$ as required.

2. The proof proceeds by induction on the derivation of $\Gamma \vdash_\omega e_0 : \tau \,\&\, \varphi \rightsquigarrow e_1$, with case analysis on the last rule used.

– Cases where the last rule is TrO-Var, TrO-Const, TrO-Fun, or TrO-SubV: These cases cannot happen since e_1 must be an extended value, which contradicts with $e_1 \longrightarrow_D e_1''$.

– Case TrO-Dcl: In this case, $e_0 = d_j\langle\!\langle e_{01} \rangle\!\rangle$ and $e_1 = \#_j(e_{11})\ \langle\rangle$ with $\Gamma \vdash_\omega e_{01} : int_{\{d_j \mapsto \omega\}} \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma(d_j) = int_{\mathbf{L}} \overset{\varphi_0}{\rightarrow}_\omega \tau$. By the assumption $e_1 \longrightarrow_D e_1''$, there are two cases to consider.
- Case where $e_1'' = \#_j(e_{11}'')\ \langle\rangle$ and $e_{11} \longrightarrow_D e_{11}''$. By the induction hypothesis, there exist e_{01}' and e_{11}' such that $e_{11}'' \longrightarrow_D^* e_{11}'$ and $e_{01} \longrightarrow_D e_{01}'$ with $\Gamma \vdash_\omega e_{01}' : int_{\{d_j \mapsto \omega\}} \,\&\, \varphi_1 \rightsquigarrow e_{11}'$. Then, the required result holds for $e_0' = d_j\langle\!\langle e_{01}' \rangle\!\rangle$ and $e_1' = \#_j(e_{11}')\langle\rangle$.

- Case where $e_{01} = n$ and $e_{11} = \langle \lambda^\omega x.d_1\langle\!\langle n \rangle\!\rangle, \ldots, \lambda^\omega x.d_k\langle\!\langle n \rangle\!\rangle \rangle$. In this case, $e_1'' = (\lambda^\omega x.d_j\langle\!\langle n \rangle\!\rangle)\langle\,\rangle$. Suppose $D(d_j) = \lambda^\omega y.e_{d_j}$. Then, the required result holds for $e_0' = e_1' = [n/y]e_{d_j}$.

– Case TrO-Fix: In this case, $e_0 = \mathbf{fix}\ x(y) = e_{01}$ and $e_1 = \mathbf{fix}\ x(y) = e_{11}$ with $\Gamma, x : \tau_1 \xrightarrow{\mathtt{nt}}_\omega \tau_2, y : \tau_1 \vdash_\omega e_{01} : \tau_2 \,\&\, \varphi \rightsquigarrow e_{11}$. e_1'' must be $\lambda^\omega y.[\mathbf{fix}\ x(y) = e_{11}/x]e_{11}$. The required result holds for $e_1' = e_1''$ and $e_0' = \lambda^\omega y.[\mathbf{fix}\ x(y) = e_{01}/x]e_{01}$.

– Case TrO-App: In this case, $e_0 = e_{01}e_{02}$ and $e_1 = e_{11}\ e_{12}$ with $\Gamma \vdash_\omega e_{01} : \tau_1 \xrightarrow{\varphi_0}_\omega \tau \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : \tau_1 \,\&\, \varphi_2 \rightsquigarrow e_{12}$. By the assumption $e_1 \longrightarrow_D e_1''$, there are three cases to consider.
  - Case where $e_1'' = e_{11}''\ e_{12}$ and $e_{11} \longrightarrow_D e_{11}''$. By the induction hypothesis, tehre exist e_{01}' and e_{11}' suc that $e_{01} \longrightarrow_D^* e_{01}'$ and $e_{11}'' \longrightarrow_D e_{11}'$ with $\Gamma \vdash_\omega e_{01}' : \tau_1 \xrightarrow{\varphi_0}_\omega \tau \,\&\, \varphi_1 \rightsquigarrow e_{11}'$. The required result holds for $e_0' = e_{01}'e_{02}$ and $e_1' = e_{11}'e_{12}$.
  - Case where $e_1'' = e_{11}e_{12}''$ and $e_{12} \longrightarrow_D e_{12}''$.
    Similar to the above case.
  - Case where $e_{11} = \lambda^\omega x.e_{13}$ and $e_1'' = [e_{12}/x]e_{13}$.
    In this case, e_{12} must be an extended value. By the transformation rules, the last rule used for deriving $\Gamma \vdash_\omega e_{01} : \tau_1 \xrightarrow{\varphi_0}_\omega \tau \,\&\, \varphi_1 \rightsquigarrow e_{11}$ must be either TrO-Abs or TrO-SubV.
    If the rule is TrO-Abs, then $e_{01} = \lambda^\omega x.e_{03}$ with $\Gamma, x : \tau_1 \vdash_\omega e_{03} : \tau \,\&\, \varphi_0 \rightsquigarrow e_{13}$. Let $e_0' = [e_{02}/x]e_{03}$ and $e_1' = e_1''$. Then, we have $e_0 \longrightarrow_D^* e_0'$ and $e_1'' \longrightarrow_D^* e_1'$. Moreover, by Lemma 15, we get $\Gamma \vdash_\omega e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$ as required.
    If the rule is TrO-SubV, then we have:

$$
\begin{array}{l}
e_{13} = \mathbf{let}\ y = (\lambda^\omega z.e_{14})\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(x)\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(y) \\
e_{01} = \lambda^\omega z.e_{04} \\
\Gamma, z : \tau_1' \vdash e_{04} : \tau' \,\&\, \varphi_0 \rightsquigarrow e_{14} \\
e_1'' = \mathbf{let}\ y = (\lambda^\omega z.e_{14})\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(y)
\end{array}
$$

    By Lemma 15, we have

$$\Gamma \vdash [e_{02}/z]e_{04} : \tau' \,\&\, \varphi_0 \rightsquigarrow [\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})/z]e_{14}.$$

    Let $e_0' = [e_{02}/y]e_{04}$. Let e_1' be $\mathbf{let}\ y = [\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})/z]e_{14}\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(y)$ if e_0' is a non-value, and $\mathbf{coerce}_{\tau' \rightsquigarrow \tau}([\mathbf{coerce}_{\tau_1 \rightsquigarrow \tau_1'}(e_{12})/z]e_{14})$ otherwise. Then, we have $e_0 \longrightarrow_D^* e_0'$ and $e_1'' \longrightarrow_D^* e_1'$ with $\Gamma \vdash e_0' : \tau \,\&\, \varphi \rightsquigarrow e_1'$ as required.

– Case TrO-Op: In this case, $e_0 = e_{01} \oplus e_{02}$ and $e_1 = e_{11} \oplus e_{12}$ with $\Gamma \vdash_\omega e_{01} : int_{\mathbf{L}} \,\&\, \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : int_{\mathbf{L}} \,\&\, \varphi_2 \rightsquigarrow e_{12}$. By the assumption $e_1 \longrightarrow_D e_1''$, there are three cases to consider.
  - Case where $e_1'' = e_{11}'' \oplus e_{12}$ and $e_{11} \longrightarrow_D e_{11}''$. By the induction hypothesis, there exist e_{01}' and e_{11}' such that $e_{01} \longrightarrow_D^* e_{01}'$ and $e_{11}'' \longrightarrow_D^* e_{11}'$ with $\Gamma \vdash_\omega e_{01}' : int_{\mathbf{L}} \,\&\, \varphi_1 \rightsquigarrow e_{11}'$. The required result holds for $e_0' = e_{01}' \oplus e_{02}$ and $e_1' = e_{11}' \oplus e_{12}$.

- Case where $e_1'' = e_{11} \oplus e_{12}''$ and $e_{12} \longrightarrow_D e_{12}''$.
  Similar to the above case.
- Case where $e_{1i} = n_i$ and $e_1'' = n_1 \underline{\oplus} n_2$.
  By the transformation rules, $e_{01} = n_1$ and $e_{02} = n_2$. The required result holds for $e_0' = e_1' = e_1''$.

– Case TrO-OpH1: In this case, $e_0 = e_{01} \oplus e_{02}$ and $e_1 = \mathbf{let}\ x_1 = e_{11}\ \mathbf{in\ let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\rangle$ with $\Gamma \vdash_\omega e_{01} : int_{p_1} \& \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : int_{p_2} \& \varphi_2 \rightsquigarrow e_{12}$. Morevoer, $\lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}$ and e_{01} is not a value. Since e_{11} is not a value, it must be the case that $e_1'' = \mathbf{let}\ x_1 = e_{11}''\ \mathbf{in\ let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\rangle$ and $e_{11} \longrightarrow_D e_{11}''$. By the induction hypothesis, there exist e_{01}' and e_{11}' such that $e_{11}'' \longrightarrow_D^* e_{11}'$ and $e_{01} \longrightarrow_D^* e_{01}'$ with $\Gamma \vdash_\omega e_{01}' : int_{p_1} \& \varphi_1 \rightsquigarrow e_{11}'$. Let $e_0' = e_{01}' \oplus e_{02}$. Let e_1' be $\mathbf{let}\ x_1 = e_{11}'\ \mathbf{in\ let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\rangle$ if e_{01}' is a non-value, and $\mathbf{let}\ x_2 = e_{12}\ \mathbf{in}\ \langle\rangle$ otherwise. Then, we have $e_0 \longrightarrow_D^* e_0'$ and $e_1'' \longrightarrow_D^* e_1'$ with $\Gamma \vdash_\omega e_0' : int_{\mathbf{H}} \& \varphi \rightsquigarrow e_1'$ as required.

– Case TrO-OpH2: In this case, $e_0 = n_1 \oplus e_{02}$ and $e_1 = \mathbf{let}\ x = e_{12}\ \mathbf{in}\ \langle\rangle$ with $\Gamma \vdash_\omega n_1 : int_{p_1} \& \varphi_1 \rightsquigarrow e_{11}$ and $\Gamma \vdash_\omega e_{02} : int_{p_2} \& \varphi_2 \rightsquigarrow e_{12}$ where $\lceil p_1 \rceil \sqcup \lceil p_2 \rceil = \mathbf{H}$. By the assumption $e_1 \longrightarrow_D e_1''$, there are two cases to consider.

- Case where $e_1'' = \mathbf{let}\ x = e_{12}''\ \mathbf{in}\ \langle\rangle$ and $e_{12} \longrightarrow_D e_{12}''$. By the induction hypothesis, there exist e_{02}' and e_{12}' such that $e_{12}'' \longrightarrow_D^* e_{12}'$ and $e_{02} \longrightarrow_D e_{02}'$ with $\Gamma \vdash_\omega e_{02}' : int_{p_2} \& \varphi_2 \rightsquigarrow e_{12}'$. The required result holds for $e_0' = n_1 \oplus e_{02}'$ and $e_1' = \mathbf{let}\ x = e_{12}'\ \mathbf{in}\ \langle\rangle$.
- Case where e_{12} is a value and $e_1'' = \langle\rangle$. By the transformation rules, e_{02} must be an integer n_2. Let $e_0' = n_1 \underline{\oplus} n_2$ and $e_1' = \langle\rangle$. Then, we have $e_0 \longrightarrow_D^* e_0'$ and $e_1'' \longrightarrow_D^* e_1'$. Moreover, by using TrO-Const and Tr-SubV, we obtain $\Gamma \vdash_\omega e_0' : int_{\mathbf{H}} \& \varphi \rightsquigarrow e_1'$ as required.

– Case TrO-If: In this case, $e_0 = \mathbf{if}\ e_{00}\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1 = \mathbf{if}\ e_{10}\ \mathbf{then}\ e_{11}\ \mathbf{else}\ e_{12}$ with $\Gamma \vdash_\omega e_{00} : int_{\mathbf{L}} \& \varphi_0 \rightsquigarrow e_{10}$ and $\Gamma \vdash_\omega e_{0i} : int_{p_i} \& \varphi_i \rightsquigarrow e_{1i} (i \in \{1, 2\})$. By the assumption $e_1 \longrightarrow_D e_1''$, there three cases to consider.

- Case where $e_1'' = \mathbf{if}\ e_{10}''\ \mathbf{then}\ e_{11}\ \mathbf{else}\ e_{12}$ and $e_{10} \longrightarrow_D e_{10}''$. By the induction hypothesis, there exist e_{00}' and e_{10}' such that $e_{10}'' \longrightarrow_D^* e_{10}'$ and $e_{00} \longrightarrow_D e_{00}'$ with $\Gamma \vdash_\omega e_{00}' : int_{\mathbf{L}} \& \varphi_0 \rightsquigarrow e_{10}'$. The required result holds for $e_0' = \mathbf{if}\ e_{00}'\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1' = \mathbf{if}\ e_{10}'\ \mathbf{then}\ e_{11}\ \mathbf{else}\ e_{12}$.
- Case where $e_{10} = n (\neq 0)$ and $e_1'' = e_{11}$. By the transformation rule, it must be the case that $e_{00} = n$. The required result holds for $e_0' = e_{01}$ and $e_1' = e_1''$.
- Case $e_{10} = 0$ and $e_1'' = e_{12}$.
  Similar to the above case.

– Case TrO-IfH: In this case, $e_0 = \mathbf{if}\ e_{00}\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e_1 = \mathbf{let}\ x = e_{10}\ \mathbf{in}\ \langle\rangle$ with $\Gamma \vdash_\omega e_{00} : int_{p_0} \& \varphi \rightsquigarrow e_{10}$ and $\Gamma \vdash_\omega e_{0i} : int_{p_i} \& \mathtt{t} \rightsquigarrow e_{1i} (i \in \{1, 2\})$, where $\lceil p_0 \rceil = \mathbf{H}$. By the assumption $e_1 \longrightarrow_D e_1''$, there are two cases to consider.

- Case where $e_{10} \longrightarrow_D e''_{10}$ and $e''_1 = \mathbf{let}\ x = e''_{10}\ \mathbf{in}\ \langle\,\rangle$.
  By the induction hypothesis, there exist e'_{00} and e'_{10} such that $e_{00} \longrightarrow^*_D e'_{00}$ and $e''_{10} \longrightarrow^*_D e'_{10}$ with $\Gamma \vdash_\omega e'_{00} : int_p \,\&\, \varphi \rightsquigarrow e'_{10}$. The required result holds for $e'_0 = \mathbf{if}\ e'_{00}\ \mathbf{then}\ e_{01}\ \mathbf{else}\ e_{02}$ and $e'_1 = \mathbf{let}\ x = e'_{10}\ \mathbf{in}\ \langle\,\rangle$.
- Case where $e_{10} = n$ and $e''_1 = \langle\,\rangle$. By the transformation rules, e_{00} must be n. Since $\Gamma \vdash_\omega e_{0i} : int_{p_i} \,\&\, \mathtt{t}$, there exists n_i such that $e_{0i} \longrightarrow^*_D n_i$. Let e'_0 be n_1 if $n \neq 0$ and n_2 otherwise. Then, the required result holds for $e'_1 = \langle\,\rangle$.

– Case TrO-Tuple: This case follows immediately from the induction hypothesis.
– Case TrO-SubE: In this case, $e_1 = \mathbf{let}\ x = e_{11}\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(x)$ with $\Gamma \vdash_\omega e_0 : \tau' \,\&\, \varphi' \rightsquigarrow e_{11}$. Since e_0 is not an extended value, neither is e_{11}. Therefore, $e''_1 = \mathbf{let}\ x = e''_{11}\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(x)$ and $e_{11} \longrightarrow_D e''_{11}$ for some e'_{11}. By the induction hypothesis, there exist e'_0 and e'_{11} such that $e_0 \longrightarrow^*_D e'_0$ and $e''_{11} \longrightarrow^*_D e'_{11}$ with $\Gamma \vdash_\omega e'_0 : \tau' \,\&\, \varphi' \rightsquigarrow e'_{11}$. Let e'_1 be $\mathbf{let}\ x = e'_{11}\ \mathbf{in}\ \mathbf{coerce}_{\tau' \rightsquigarrow \tau}(x)$ if e'_0 is a non-value, and be $\mathbf{coerce}_{\tau' \rightsquigarrow \tau}(e'_{11})$ otherwise. Then we have $e''_1 \longrightarrow^*_D e'_1$ and $\Gamma \vdash e'_0 : \tau \,\&\, \varphi \rightsquigarrow e'_1$ as required.

□

Corollary 1. *Let D be a declassification environment. If $\vdash_\omega D : \Gamma$ and $\Gamma \vdash_\omega e_0 : int_{\mathbf{L}} \,\&\, \varphi \rightsquigarrow e_1$, then $e_0 \longrightarrow^*_D 0$ if and only if $e_1 \longrightarrow^*_D 0$*

Proof. Suppose $e_0 \longrightarrow^*_D 0$. By Lemma 16, there exists e'_1 such that $e_1 \longrightarrow^*_D e'_1$ and $\Gamma \vdash_\omega 0 : int_{\mathbf{L}} \,\&\, \varphi \rightsquigarrow e'_1$. By the transformation rule, e'_1 must be 0.

The converse is similar.

We now prove Lemma 14.

Proof of Lemma 14 Let D be a declassification environment. Suppose also that $\vdash D : \Gamma$ and $\Gamma \vdash e : int_{\mathbf{L}} \,\&\, \varphi \rightsquigarrow e'$. By Lemma 12, we have $\Gamma \vdash e : int_{\mathbf{L}}$ and $[\![\Gamma]\!]_\vdash e' : [\![int_{\mathbf{L}}]\!] \,\&\, \varphi$. Let D^ω and e^ω be the declassification environment and expression obtained by replacing all the uses in D and e with ω. By Theorem 1, the reduction of $\langle D, e\rangle$ and $\langle D, e'\rangle$ cannot get stuck. So, $\longrightarrow$ coincides with the linearity-insensitive reduction relation $\longrightarrow_{D^\omega}$, i.e., $\langle D, e\rangle \Downarrow 0$ if and only if $e^\omega \longrightarrow^*_{D^\omega} 0$; and $\langle D, e'\rangle \Downarrow 0$ if and only if $e'^\omega \longrightarrow^*_{D^\omega} 0$. Therefore, we have:

$$
\begin{aligned}
& \langle D, e\rangle \Downarrow 0 \\
\iff & e^\omega \longrightarrow^*_{D^\omega} 0 \\
\iff & e'^\omega \longrightarrow^*_{D^\omega} 0 \text{ (by Corollary 1)} \\
\iff & \langle D, e'\rangle \Downarrow 0
\end{aligned}
$$

□