Type-Based Information Flow Analysis for the Pi-Calculus

Naoki Kobayashi
Tokyo Institute of Technology
email:kobayasi@cs.titech.ac.jp

Abstract

We propose a new type system for information flow analysis for the π-calculus. Previous type-based information flow analyses for concurrent programs have imposed rather heavy restrictions on usage of communication/synchronization primitives. In contrast, our type system is general and expressive, so that information flow analysis for concurrent programs using a variety of concurrency primitives can be uniformly discussed through the π-calculus. The generality of our type system has also enabled development of a clear proof of type soundness and a type inference algorithm.

1 Introduction

Information flow analysis is a static program analysis to check that a program does not leak information about secret data. Since Denning and Denning's work [5], information flow analysis has been studied for various programming languages, including imperative languages [5, 31], functional languages [8, 26], low-level languages [19, 32], and concurrent languages [11, 12, 25, 29, 33, 27].

Previous information flow analyses (especially, automated ones) for concurrent languages have not been quite satisfactory. Most of them either do not deal with communication/synchronization primitives (e.g., [29])[1] or impose rather heavy restriction on usage of communication or synchronization primitives. For example, consider the four threads in Figure 1, which use lock primitives. Suppose that the lock `X` can be accessed by only privileged principals, while the lock `Y` can be accessed by any (possibly malicious) principals. Suppose also that the variable `secret` holds a secret boolean value while the variable `public` can be accessed by anyone, and that `P` and `R` accesses secret data guarded by lock X, and `Q` accesses data guarded by X and Y. Then, is it safe to execute threads A and B concurrently? What if B is replaced by C or D?[2] What if lock primitives are replaced by send/receive primitives or barrier synchronization primitives? As far as the author knows, the previous information flow analyses for concurrent languages cannot satisfactorily deal with such programs.

To deal with various synchronization/concurrency primitives in a uniform manner, we use the π-calculus as the target language and develop a type system for information flow analysis. The type system ensures that well-typed processes do not leak secret information, so that the problem of information flow analysis is reduced to that of type checking or inference.

[1] One may argue that synchronization primitives can be expressed in terms of shared memory primitives by using, for example, Peterson's algorithm. Type-based information flow analyses do not, however, work well for such encodings: As explained below, to analyze information flow, it is important to analyze whether each communication or synchronization succeeds or not. Since synchronization algorithms based on shared memory primitives inspect the values of shared variables (like flag variables), type systems cannot statically infer whether each synchronization succeeds.

[2] The answer is: The former is unsafe since if a malicious thread holds lock Y forever, it can get information about `secret` by checking whether `public:=true` is executed. On the other hand, the latter ones are (arguably) safe provided that the lock `X` is carefully used at other places so that `X` is not held forever.

Thread A	**Thread B**	**Thread C**	**Thread D**
`if(!secret)`	`lock(X);`	`lock(Y);`	`lock(X);`
`{lock(X);`	`lock(Y);`	`lock(X);`	`R;`
`P;`	`Q;`	`Q;`	`unlock(X);`
`unlock(X)};`	`unlock(Y);`	`unlock(X);`	`public := false`
`public := true`	`unlock(X);`	`unlock(Y);`	

Figure 1: Threads using lock-primitives

Type systems for information flow analysis for the π-calculus and similar calculi have been studied by Honda and Yoshida [11, 12], Pottier [25], Hennessy and Riely [10, 9], and Zdancewic and Myers [33]. Pottier's type system [25] and Hennessy and Riely [10, 9] are, however, not expressive enough: Let us consider a process $x(\,).\,P$, which waits to receive a null tuple on channel x and then behaves like P. In Pottier's type system, if x is a secret channel, then P can communicate via only secret channels (since if P performs communication on non-secret channels, it may reveal information that someone has sent a message on the secret channel x). So, none of the threads in Figure 1 can be typed (if lock primitives are encoded into communication primitives in the π-calculus). Honda and Yoshida [12] and Zdancewic and Myers [33] overcome this problem by allowing P to communicate via non-secret channels if the input on x always succeeds. For that purpose, they introduced special kinds of communication channels, and constructed a type system guaranteeing that communications on such channels always succeed, so that $x(\,).\,P$ is allowed as long as x is such a channel, even if x is secret and P communicate through non-secret channels. A problem of those type systems [12, 33] is, however, that they can deal with only specific kinds of channels (such as linear channels). For example, they cannot deal with processes using locks, like those in Figure 1. Another problem of Honda and Yoshida's type system [12] is that there seems to be no reasonable type inference algorithm (which works as an algorithm for information flow analysis) for their type system: Since they introduce a separate typing rule for each kind of channel, a programmer at least needs to explicitly declare the kind of each channel to enable type inference.

We overcome the problems mentioned above by using ideas of our previous type systems for deadlock/livelock-freedom [15, 18, 30]. As in Honda and Yoshida's type system [12], our type system for information flow analysis accepts $x(\,).\,P$ as long as it is statically known that the input on x succeeds, even if x is secret and P communicates over non-secret channels. The main difference is that our type system can deal with arbitrary usage of channels in a very uniform manner, so that programs using various concurrency primitives (including locks) can be encoded into the π-calculus and analyzed by using our type system. The uniform treatment of channels also enabled us to develop a clear proof of type soundness (which says that well-typed processes do not leak secret information) and a type inference algorithm (i.e. an information flow analysis algorithm). The type inference algorithm is sound and complete with respect to a slightly less expressive variant of the type system (which is still expressive enough to deal with the above programs).

In addition to the refinement of information flow analysis for the π-calculus, the present work can also be considered a refinement of our previous type system for lock-freedom [15]. The previous type system [15] could guarantee that a certain communication eventually succeeds, but required explicit type annotation. By using our new type system, we can automatically analyze whether each communication eventually succeeds or not.

The rest of this paper is structured as follows. Section 2 introduces the target language of our type-based information flow analysis. Section 3 presents our type system for information flow analysis, and Section 4 proves its soundness. Section 5 describes a type inference algorithm. Section 6 discusses limitations of our type system and Section 7 discusses related work. Section 8 concludes. Proofs are found

in Appendix.

2 Target Language

This section introduces the target language of our type-based information flow analysis. The language is a subset of the polyadic π-calculus [21].

2.1 Syntax

We first introduce secrecy levels, which denote the degree of secrecy of information about data and processes. For the sake of simplicity, we only consider two secrecy levels: **H**, which describes secret information (i.e., information that should be kept to privileged principals), and **L**, which describes non-secret information (i.e., information that can be revealed to any principals).

Definition 2.1 [secrecy levels]: The set of *secrecy levels* is $\{\mathbf{H}, \mathbf{L}\}$. The binary relation $\sqsubseteq$ on secrecy levels is the total order defined by $\mathbf{L} \sqsubseteq \mathbf{H}$.

We use a meta-variable l for a secrecy level.

Definition 2.2 [processes]: The set of processes, ranged over by P, is defined by:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \mid \overline{x}\langle v_1, \ldots, v_n \rangle . P \mid x(y_1, \ldots, y_n) . P \\ & & \mid (P \mid Q) \mid *P \mid (\nu x : \xi)\, P \mid \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q \\ v & ::= & \mathit{true}^l \mid \mathit{false}^l \mid \star \mid x \end{array}$$

Here, x and y_i range over a countably infinite set **Var** of variables. ξ ranges over the set of core channel types (which is defined later in Section 3).

Notation 2.3: The prefix $x(y_1, \ldots, y_n)$ binds variables $y_1, \ldots, y_n$ and $(\nu x : \xi)$ binds x. As usual, we identify processes up to α-conversions (renaming of bound variables), and assume that α-conversions are implicitly applied so that bound variables are always different from each other and from free variables. We write $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]P$ for the process obtained by replacing all the free occurrences of $x_1, \ldots, x_n$ in P with $v_1, \ldots, v_n$. We write $\tilde{x}$ for a sequence of variables $x_1, \ldots, x_n$. We abbreviate $[x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]$ and $(\nu x_1 : \xi_1) \cdots (\nu x_n : \xi_n)$ to $[\tilde{x} \mapsto \tilde{v}]$ and $(\nu \tilde{x} : \tilde{\xi})$ respectively. We often omit $\mathbf{0}$ and write $\overline{x}\langle \tilde{v} \rangle$ and $x(\tilde{y})$ for $\overline{x}\langle \tilde{v} \rangle . \mathbf{0}$ and $x(\tilde{y}) . \mathbf{0}$ respectively.

We assume that prefixes ($\overline{x}\langle \tilde{v} \rangle$, $x(\tilde{y})$, (νx), and $*$) bind tighter than the parallel composition operator $\mid$, so that $\overline{x}\langle \tilde{y} \rangle . P \mid Q$ means $(\overline{x}\langle \tilde{y} \rangle . P) \mid Q$, not $\overline{x}\langle \tilde{y} \rangle . (P \mid Q)$.

Process $\mathbf{0}$ does nothing. Process $\overline{x}\langle \tilde{v} \rangle . P$ sends a tuple $\langle \tilde{v} \rangle$ on x and then (after the tuple is received by some process) behaves like P. Each v_i is a boolean (true^l or false^l), the unit value ($\star$) (the element of a singleton set), or a variable. A communication channel is represented by a free variable or a variable bound by $(\nu x : \xi)$.

Process $x(\tilde{y}) . P$ waits to receive a tuple $\langle \tilde{v} \rangle$ on x and then behaves like $[\tilde{y} \mapsto \tilde{v}]P$. $P \mid Q$ represents concurrent execution of P and Q. $*P$ represents infinitely many copies of the process P running in parallel, and $(\nu x : \xi)\, P$ denotes a process that creates a fresh communication channel x and then behaves like P. The core channel type ξ is attached just for technical convenience (in proving soundness of the type system); It does not affect the operational semantics. A programmer need not specify core channel types, since a type inference algorithm can recover it. We often omit ξ when it is not important. **if** v **then** P **else** Q behaves like P if v is true^l and behaves like Q if v is false^l; otherwise it is blocked forever.

The secrecy level l attached to a boolean value (true^l or false^l) expresses the degree of secrecy of the value. For example, information about $\mathit{true}^{\mathbf{H}}$ should not be revealed to non-privileged principals. On the other hand, no secrecy level is attached to $\star$, since it carries no information. The secrecy level of a communication channel may be specified in the core channel type ξ of $(\nu x : \xi)$. Those annotations of secrecy levels (for values and channels) are only used for a programmer to declare which value should be hidden to non-privileged principals. The programmer can omit them if they are unnecessary, since the type inference algorithm described in Section 5 can recover them.

2.2 Operational Semantics

Following the standard reduction semantics for the π-calculus, we define the operational semantics using a structural relation $P \preceq Q$, and a reduction relation $P \longrightarrow Q$. The former relation means that P can be restructured to Q by using the commutativity and associativity laws on $|$, etc. The latter relation means that P is reduced to Q by one communication on a channel. Differences from the standard reduction semantics are that $\preceq$ is not symmetric, and that we include reductions on if-expressions in $\preceq$ rather than $\longrightarrow$ (so that **if** true^l **then** P **else** $Q \preceq P$).

Definition 2.4: The *structural preorder* $\preceq$ is the least reflexive and transitive relation closed under the following rules ($P \equiv Q$ denotes $(P \preceq Q) \wedge (Q \preceq P)$):

$$P \equiv P \mid \mathbf{0} \qquad \text{(S-Zero1)}$$

$$\mathbf{0} \equiv *\mathbf{0} \qquad \text{(S-Zero2)}$$

$$\mathbf{0} \equiv (\nu x : \xi)\, \mathbf{0} \qquad \text{(S-Zero3)}$$

$$P \mid Q \equiv Q \mid P \qquad \text{(S-Commut)}$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad \text{(S-Assoc)}$$

$$(\nu x : \xi)\, P \mid Q \equiv (\nu x : \xi)\, (P \mid Q) \quad (\text{if } x \text{ is not free in } Q) \qquad \text{(S-New)}$$

$$(\nu x : \xi_1)\, (\nu y : \xi_2)\, P \equiv (\nu y : \xi_2)\, (\nu x : \xi_1)\, P \qquad \text{(S-Swap)}$$

$$\textbf{if } \mathit{true}^l \textbf{ then } P \textbf{ else } Q \preceq P \qquad \text{(S-IfT)}$$

$$\textbf{if } \mathit{false}^l \textbf{ then } P \textbf{ else } Q \preceq Q \qquad \text{(S-IfF)}$$

$$*P \preceq *P \mid P \qquad \text{(S-Rep)}$$

$$\frac{P \preceq P'}{P \mid Q \preceq P' \mid Q} \qquad \text{(S-Par)}$$

$$\frac{P \preceq Q}{(\nu x : \xi)\, P \preceq (\nu x : \xi)\, Q} \qquad \text{(S-CNew)}$$

Definition 2.5: The reduction relation $\longrightarrow$ is the least relation closed under the following rules:

$$\overline{x}\langle \tilde{v} \rangle . P \mid x(\tilde{y}) . Q \longrightarrow P \mid [\tilde{y} \mapsto \tilde{v}]Q \qquad \text{(R-Com)}$$

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R} \qquad \text{(R-Par)}$$

$$\frac{P \longrightarrow Q}{(\nu x : \xi)\, P \longrightarrow (\nu x : \xi)\, Q} \qquad \text{(R-New)}$$

$$\frac{P \preceq P' \qquad P' \longrightarrow Q' \qquad Q' \preceq Q}{P \longrightarrow Q} \qquad \text{(R-SP)}$$

We write $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

2.3 Examples

As given above, the basic π-calculus has only a few primitives, but various mechanisms present in real programming languages can be easily encoded [22, 28]. We give below some examples of such encodings, which will be used afterwards. For the sake of clarity, we regard integers and operations on them as primitives below.

Example 2.6: The process $*succ(n, r).\, \overline{r}\langle n+1 \rangle$ works as a function server computing the successor of an integer. It receives a pair consisting of an integer n and a channel r, and sends $n+1$ on channel r.

Example 2.7 [locks]: A lock (a binary semaphore) can be implemented by using a channel that holds at most one value at any moment. We can regard the presence of a value in the channel as the unlocked state, and the absence of a value as the locked state. Then, creation of a new lock `let x = newlock() in P` is encoded into a process: $(\nu x)\,(\overline{x}\langle\,\rangle \mid P)$. Lock/unlock operations `lock(x);P` and `unlock(x);P` can be encoded into processes $x(\,).\, P$ and $\overline{x}\langle\,\rangle \mid P$ respectively. (Note that the unlock operation is encoded into an asynchronous output since the unlock operation is a non-blocking operation.) Thus, Thread B in Figure 1 is encoded into $x(\,).\, y(\,).\, (\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$. Here, we have omitted the part "`Q`" in the thread.

Example 2.8 [shared variables]: Shared variables can also be implemented by using a channel that holds at most one value at any moment. Creation of a shared variable `let x=newref v in P`, the read operation `let y=!x in P`, and the write operation `x:=v; P` can be encoded respectively into: $(\nu x)\,(\overline{x}\langle v \rangle \mid P)$, $x(y).\, (\overline{x}\langle y \rangle \mid P)$, and $x(y).\, (\overline{x}\langle v \rangle \mid P)$.

Example 2.9 [sequencing]: A sequential execution `P;Q` can also be easily encoded by using the idea of continuation-passing. Let P be a process that simulates `P` and signals its termination to a channel c, and Q be a process that simulates `Q`. Then, `P;Q` is encoded into $(\nu c)\,(P \mid c(\,).\, Q)$. Thus, Thread A in Figure 1 (with the part "`P`" being omitted) is expressed as the following process A:

$$\begin{array}{l} (\nu c)\, secret(b).\, (\overline{secret}\langle b \rangle \\ \qquad \mid \textbf{if } b \textbf{ then } x(\,).\, (\overline{x}\langle\,\rangle \mid \overline{c}\langle\,\rangle) \textbf{ else } \overline{c}\langle\,\rangle \\ \qquad \mid c(\,).\, public(z)\,.\, \overline{public}\langle true^{\mathbf{L}} \rangle\,) \end{array}$$

Here, channel c is used to signal the termination of the if-expression.

Example 2.10: The **cobegin/coend** statement: `cobegin P1 | ... | Pn coend` (which executes `P1`,..., `Pn` concurrently) can also be easily encoded. Let P_i be a process that simulates `P1` and signals its termination to c_i. Then, `cobegin P1 | ... | Pn coend; Q` can be encoded into a process: $(\nu c_1) \cdots (\nu c_n)\,(P_1 \mid \cdots \mid P_n \mid c_1(\,). \cdots c_n(\,).\, Q)$.

This construct can be freely combined with other communication/synchronization primitives. Consider the following program:

```
cobegin
    (x:=true; send(c, nil))
  | let _=receive(c) in y:=!x
corend;
w := false
```

Here, `send` and `receive` are commands to synchronously send/receive values. The first thread of the `cobegin` is a producer, which writes `true` to the variable `x` and notify that it is ready, and the second one waits for the notification and reads the variable `x`. It can be encoded into:

$$\begin{array}{l}(\nu c_1)\,(\nu c_2)\,(x(z).\,(\overline{x}\langle \mathit{true}^{\mathbf{H}}\rangle \mid \overline{c}\langle\,\rangle.\,\overline{c_1}\langle\,\rangle)\\ \qquad \mid c(y).\,x(z).\,(\overline{x}\langle z\rangle \mid y(u).\,(\overline{y}\langle z\rangle \mid \overline{c_2}\langle\,\rangle)))\\ \qquad \mid c_1(\,).\,c_2(\,).\,w(z)\,.\,\overline{w}\langle \mathit{false}^{\mathbf{L}}\rangle)\end{array}$$

Example 2.11: Concurrent objects can be easily expressed in the π-calculus [24, 20]. The following process implements a bank account object:

$$\begin{array}{l}(\nu s)\,(\overline{s}\langle 100^{\mathbf{H}}\rangle\\ \quad \mid *\mathit{withdraw}(\mathit{amount}, r).\,s(x).\\ \qquad\quad \textbf{if } x \geq \mathit{amount} \textbf{ then } (\overline{r}\langle \mathit{true}^{\mathbf{H}}\rangle \mid \overline{s}\langle x - \mathit{amount}\rangle)\\ \qquad\quad \textbf{else } (\overline{r}\langle \mathit{false}^{\mathbf{H}}\rangle \mid \overline{s}\langle x\rangle)\\ \quad \mid *\mathit{deposit}(\mathit{amount}, r).\,s(x).\,(\overline{r}\langle x + \mathit{amount}\rangle \mid \overline{s}\langle x + \mathit{amount}\rangle)\\ \quad \mid *\mathit{transfer}(\mathit{amount}, r).\,s(x).\,(\overline{r}\langle\,\rangle \mid \overline{s}\langle x + \mathit{amount}\rangle))\end{array}$$

It stores the current balance in the channel s, and handles three kinds of requests through channels *withdraw*, *deposit*, and *transfer*. Upon receiving a request on *withdraw*, the process checks whether the current balance is enough and replies whether the withdraw operation has succeeded or not. Upon receiving a request on *deposit*, the process updates the current balance and sends the new balance as an acknowledgment. Upon receiving a request on *transfer* (for transferring the specified amount of money to this account), the process updates the current balance and sends an acknowledgment on r. The channels *withdraw* and *deposit* are secret, so that only a privileged person can send requests, while the channel *transfer* (for sending a request for transferring money to this account) can be accessed by anyone. The current balance should also be kept secret, so that 100 is annotated with **H**. The type system presented in the next section can guarantee that one cannot obtain any information about the current balance through the public channel *transfer* (see Example 3.28). Note that the previous type systems for information flow-analysis for the π-calculus and similar calculi [11, 12, 25, 33] cannot accept the process above, since the sub-process $*\mathit{transfer}(\mathit{amount}, r).\,\cdots$ sends a message through a non-secret channel r after receiving a value through the secret channel s.

3 Type System for Information Flow Analysis

This section introduces a type system for information flow analysis. The type system guarantees that any well-typed process does not leak secret information, so that the problem of checking whether a process leaks secret information is reduced to the problem of type inference. We first explain main ideas of the type system in Subsection 3.1, and then introduce formal definitions in later subsections.

3.1 Overview

We explain ideas of the type system informally in three steps.

3.1.1 Extending types with secrecy levels

As in other type systems for information flow analysis, we extend the usual types with secrecy levels. For example, the type **bool** of booleans is refined to $\mathbf{bool}^{\mathbf{H}}$ and $\mathbf{bool}^{\mathbf{L}}$: The former is the type of secret booleans and the latter is the type of non-secret booleans.

Since information about values may be propagated through the behavior of processes, we also need to consider secrecy levels of *channels* and *processes*. For example, if a process $\mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ \overline{x}\langle\,\rangle\ \mathbf{else}\ \mathbf{0}$ is executed, information about the boolean $\mathit{true}^{\mathbf{H}}$ is propagated through information about whether a message is sent on x or not. Moreover, if a process $x(\,).\,P$ is executed in parallel, the information is further propagated through information about whether the process P is executed. To keep track of this kind of information flow, we let the secrecy level of a channel express the degree of secrecy of information about what communication takes place on the channel (e.g., whether some message is sent on the channel), and let the secrecy level of a process express the degree of secrecy of information about whether the process is executed. (The latter corresponds to the secrecy level of a program counter in information flow analysis for imperative languages [5].) In the example above, we consider that the secrecy levels of the channel x and the process P are also $\mathbf{H}$.

We write $\langle\tau\rangle^{l}$ for the type of a channel of secrecy level l that is used for transmitting values of type τ The secrecy level of a channel should not be confused with the secrecy level of values sent on the channel. For example, if a channel has type $\langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{L}}$, then a non-privileged principal may obtain information about whether some message is sent along the channel although he or she cannot obtain information about the contents of the message.

A type judgment for a process is of the form $x_1:\tau_1,\ldots,x_n:\tau_n \vdash_l P$, where $\tilde{\tau}$ are extended types and l denotes the secrecy level of the behavior of P. If l is $\mathbf{H}$, the behavior of P is only observable to privileged principals. For example, $x:\langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}} \vdash_{\mathbf{H}} \overline{x}\langle\mathit{true}^{\mathbf{H}}\rangle$ is valid, but $x:\langle\mathbf{bool}^{\mathbf{L}}\rangle^{\mathbf{L}} \vdash_{\mathbf{H}} \overline{x}\langle\mathit{true}^{\mathbf{L}}\rangle$ is invalid, since the latter process sends a value on a non-secret channel, so that its behavior is observable to any principals.

Based on the intuition above, it would not be difficult to understand the following rule for if-expressions:

$$\frac{\Gamma \vdash v:\mathbf{bool}^{l_1} \qquad \Gamma \vdash_{l_2} P \qquad \Gamma \vdash_{l_2} Q \qquad l_1 \sqsubseteq l_2}{\Gamma \vdash_{l_2} \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q}$$

Since one can infer the value of v by observing whether P or Q is executed, the secrecy level l_2 of P and Q must be greater than or equal to the secrecy level l_1 of v.

Based on a similar intuition, we can obtain the following rule for an input process $x(y).\,P$:

$$\frac{\Gamma, x:\langle\tau\rangle^{l_1}, y:\tau \vdash_{l_2} P \qquad l \sqsubseteq l_1 \sqsubseteq l_2}{\Gamma, x:\langle\tau\rangle^{l_1} \vdash_{l} x(y).\,P} \qquad \text{(In-Naive)}$$

Since one can infer whether some process sends a value on x by observing whether P is executed or not, the rule imposes the condition $l_1 \sqsubseteq l_2$.

3.1.2 Extending channels types with usage expressions

The rule In-Naive given above is actually too naive. Because of the condition $l_1 \sqsubseteq l_2$, once a process performs an input on a secret channel, it can no longer perform communications on non-secret channels. For example, the process A in Example 2.9 cannot be typed: Since n is a secret value, both channels x and c must be secret, so that $c(\,).\,\mathit{public}(z).\,\overline{\mathit{public}}\langle\mathit{true}^{\mathbf{L}}\rangle$ cannot be typed. Actually, however, as long as the input on x always succeeds, the input on c also succeeds, so that the process does not leak any information about n.

Based on the above observation, we want to replace rule IN-NAIVE with something like:[3]

$$\frac{\Gamma, x : \langle \tau \rangle^{l_1}, y : \tau \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad l_1 \sqsubseteq l_2 \text{ if the input on } x \text{ may not succeed}}{\Gamma, x : \langle \tau \rangle^{l_1} \vdash_l x(y).\, P} \qquad \text{(IN-IDEAL)}$$

What remains to do is to replace the sentence "if the input on x may not succeed" with a statically verifiable condition. For this purpose, we use the idea of type systems for lock-freedom [15, 18, 30]. The idea is to extend channel types with *usage expressions* (usages, in short) [30, 18, 15], which specify how each channel should be used.

Usages are constructed from I, denoting an input action, and O, denoting an output action, by using sequential composition (.), parallel composition (|), etc. For example, usage $I.O$ describes a channel that should be first used once for input *and then* used once for output. So, if x has usage $I.O$, the process $x(\,).\,\overline{x}\langle\,\rangle$ is valid but the processes $\overline{x}\langle\,\rangle.\,x(\,)$ and $x(\,).\,(\overline{x}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ are invalid. Usage $I \mid O$ describes a channel that should be used once for input and once for output *in parallel*. So, if x has usage $I \mid O$, then the process $x(\,) \mid \overline{x}\langle\,\rangle$ is valid. A channel type is annotated with a usage and written $\langle \tau \rangle^l / U$, which describes a channel that is used according to usage U.

Typing rules are extended to take into account usage information by using the idea of linear types [17]. For example, the rule for parallel composition is:

$$\frac{\Gamma_1 \vdash_l P_1 \qquad \Gamma_2 \vdash_l P_2}{\Gamma_1 \mid \Gamma_2 \vdash_l P_1 \mid P_2}$$

Here, $\Gamma_1 \mid \Gamma_2$ is the type environment obtained by combining usages of each variable in Γ_1 and Γ_2 with $\mid$. For example, from $x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / I \vdash_{\mathbf{H}} P_1$ and $x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O \vdash_{\mathbf{H}} P_2$ (which mean that x is used once for input in P_1 and once for output in P_2), we can derive $x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / (I \mid O) \vdash_{\mathbf{H}} P_1 \mid P_2$, which means that x is used once for input and once for output in parallel. The rule for input processes would be replaced by (for the moment, we forget conditions on secrecy levels):

$$\frac{\Gamma, x : \langle \tau \rangle^{l_1} / U, y : \tau \vdash_{l_2} P}{\Gamma, x : \langle \tau \rangle^{l_1} / I.U \vdash_l x(y).\, P} \qquad \text{(IN-WITH-USAGE)}$$

Here, the usage of x in the conclusion captures the fact that x is first used for input and then used according to U in P.

From the usage part of a channel type, we can obtain some information about whether communications succeed or not. For example, in the process $(\nu x)\, x(\,).\,\overline{y}\langle\,\rangle$, the usage of x is expressed by I. So, we know that x is used once for input but never used for output, so that the input never succeeds.

3.1.3 Refining usages with obligation/capability levels

Usage expressions explained above are not sufficient for the purpose of checking that certain communications *must* succeed, since they can express only channel-wise communication behavior of processes, not inter-channel dependencies. For example consider the process $x(\,).\,\overline{y}\langle\,\rangle \mid y(\,).\,\overline{x}\langle\,\rangle$. Usage of x and y can be expressed by $I \mid O$, but communications on x and y never succeed because of a deadlock. The problem of the above process can be explained as follows: In order for the input on x to succeed, the righthand process has an *obligation* to do an output on x. Before fulfilling the obligation, however, the righthand process is claiming a *capability* to successfully complete the input on y. In order for the input on y to succeed, the lefthand process has an *obligation* to do an output on y, but before fulfilling the obligation, the lefthand

[3] Actually, additional subtle conditions are required in order for this rule to be valid: See the first paragraph in Section 6.

process is claiming a capability to successfully complete the output on x. Thus, both processes claim their capabilities before fulfilling the obligations, so that a deadlock occurs.

In order to avoid this kind of circular dependency, we associate each I and O in usages with an *obligation level* t_o and a *capability level* t_c, and write $I^{t_o}_{t_c}$ and $O^{t_o}_{t_c}$. Obligation levels and capability levels range over the set consisting of natural numbers and ∞. The obligation level expresses the degree of an obligation to do an action, while the capability level expresses the degree of a capability to successfully complete an action. More precisely, obligation and capability levels control the behavior of processes through the following rules:

Rule A If a process holds an obligation of level n to do some action, then a process can exercise only a finite number of capabilities whose levels are less than n before fulfilling the obligation. For example, if the usage of x is O^1_3 and the usage of y is O^2_0, the process $\overline{y}\langle\,\rangle.\,\overline{x}\langle\,\rangle$ is allowed, but $\overline{x}\langle\,\rangle.\,\overline{y}\langle\,\rangle$ is not allowed.

Rule B If a process holds a capability of level n to perform an action, there must exist another process that holds an obligation to do its co-action whose level is less than or equal to n.

These rules ensure that every action of a finite capability level will eventually succeed. For example, consider the process $x(\,).\,\overline{y}\langle\,\rangle \mid y(\,) \mid \overline{x}\langle\,\rangle$. We can assign usages $I^0_0 \mid O^0_0$ and $I^0_1 \mid O^1_0$ to x and y, so that the above rules are satisfied. From the usages, we can infer that communications on x and y succeed.

On the other hand, for the deadlocked process $x(\,).\,\overline{y}\langle\,\rangle \mid y(\,).\,\overline{x}\langle\,\rangle$, there is no way to assign a finite capability level to the input on x: Let usages of x and y be $I^{t_1}_{t_2} \mid O^{t_3}_{t_4}$ and $I^{t_5}_{t_6} \mid O^{t_7}_{t_8}$. Rule A requires $t_2 < t_7$ and $t_6 < t_3$. On the other hand, Rule B requires the following conditions:

$$t_1 \le t_4 \qquad t_3 \le t_2 \qquad t_5 \le t_8 \qquad t_7 \le t_6.$$

So, we obtain the constraint $t_2 < t_7 \le t_6 < t_3 \le t_2$, so that the capability level t_2 cannot be finite.

The typing rule for input processes is now refined as follows.

$$\frac{\Gamma, x : \langle\tau\rangle^{l_1}/U, y : \tau \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c + 1 \le ob(\Gamma) \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\Gamma, x : \langle\tau\rangle^{l_1}/I^{t_o}_{t_c}.U \vdash_l x(y).\,P} \qquad \text{(In-Refined)}$$

The condition $t_c + 1 \le ob(\Gamma)$ enforces Rule A. (Here, $ob(\Gamma)$ denotes the least obligation level contained in Γ.) The statement "if the input on x may not succeed" has now been replaced by a statically verifiable condition $t_c = \infty$.

Rule B is enforced by the following typing rule for ν-prefix:

$$\frac{\Gamma, x : \langle\tau\rangle^{l_1}/U \vdash_{l_2} P \qquad rel(U)}{\Gamma \vdash_{l_2} (\nu x)\,P} \qquad \text{(New)}$$

The condition $rel(U)$ means that for any usage of the form $I^{t_o}_{t_c}.U_1$ in U, there is a corresponding usage of the form $O^{t'_o}_{t'_c}.U_2$ such that $t'_o \le t_c$ (and a similar condition for $O^{t_o}_{t_c}.U_1$ in U). For example, in order for $rel(I^{t_1}_{t_2} \mid O^{t_3}_{t_4})$ to hold, it must be the case that $t_1 \le t_4$ and $t_3 \le t_2$.

A part of the requirement of Rule A, that only a *finite* number of capabilities can be exercised before an obligation is fulfilled, is enforced by the following rule for output processes (we omit the continuation part for the sake of simplicity):

$$\frac{t_c + 1 \le ob(y : \uparrow\tau) \qquad l \sqsubseteq l_1}{x : \langle\tau\rangle^{l_1}/O^{t_o}_{t_c}.U, y : \uparrow\tau \vdash_l \overline{x}\langle y\rangle} \qquad \text{(Out)}$$

Usages	Interpretation
$\mathbf{0}$	Cannot be used at all
$I^{t_o}_{t_c}.U$	Used once for input, and then used according to U
$O^{t_o}_{t_c}.U$	Used once for output, and then used according to U
$U_1 \mid U_2$	Used according to U_1 by one process and used according to U_2 by another process
$*U$	Used according to U by infinitely many processes
$\uparrow^{(t_1,t_2)}U$	The same as U, except that input and output obligation levels are raised to t_1 and t_2 respectively.
$U_1 \,\&\, U_2$	Used according to either U_1 or U_2
ρ	Usage variable (used in combination with recursive usages below)
$\mu\rho.U$	Recursively used according to $[\rho \mapsto \mu\rho.U]U$.

Table 1: Meaning of Usage Expressions

Here, $\uparrow\tau$ is the type obtained by increasing obligation levels of τ by one. For example, $\uparrow(\langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/O^1_2) = \langle\mathbf{bool}^{\mathbf{H}}\rangle^{\mathbf{H}}/O^2_2$. This enforces that when an obligation is delegated to another process (by sending it through a channel), the level of the delegated obligation is less than the current obligation level, so that an obligation of a finite level cannot be infinitely delegated (since the level of the obligation eventually reaches 0). For example, consider the process

$$\overline{x_1}\langle y\rangle \mid x_1(z).\,\overline{x_2}\langle z\rangle \mid x_2(z).\,P.$$

Suppose that the sub-process $\overline{x_1}\langle y\rangle$ initially holds an obligation of level 2 to do an output on y. When y is sent through x_1, the obligation level becomes 1, and when it is further sent through x_2, the obligation level becomes 0, so that the process P must use y immediately. In the process $*x(z).\,\overline{x}\langle z\rangle \mid \overline{x}\langle y\rangle$, which forwards y forever, y's obligation level must be ∞.

3.2 Usages

3.2.1 Syntax

Now we introduce the formal syntax of usages.

Definition 3.1 [usages]: The set $\mathcal{U}$ of *usages*, ranged over by U, is given by the following syntax.

$$\begin{array}{l} U ::= \mathbf{0} \mid \alpha^{t_1}_{t_2}.U \mid (U_1 \mid U_2) \mid *U \mid \uparrow^{(t_1,t_2)}U \mid U_1 \,\&\, U_2 \mid \rho \mid \mu\rho.U \\ \alpha ::= I \mid O \\ t_1, t_2 \in \mathbf{Nat} \cup \{\infty\} \end{array}$$

Here, **Nat** is the set of natural numbers.

We often omit $\mathbf{0}$ and write $\alpha^{t_1}_{t_2}$ for $\alpha^{t_1}_{t_2}.\mathbf{0}$. We extend the usual binary relation $\leq$ on **Nat** to that on $\mathbf{Nat} \cup \{\infty\}$ by $\forall t \in \mathbf{Nat} \cup \{\infty\}.t \leq \infty$. We write $\mathbf{min}(x_1, \ldots, x_n)$ for the least element of $\{x_1, \ldots, x_n\}$ (∞ if $n = 0$) with respect to $\leq$ and write $\mathbf{max}(x_1, \ldots, x_n)$ for the greatest element of $\{x_1, \ldots, x_n\}$ (0 if $n = 0$). We assume that $\mu\rho$ binds ρ. We write $[\rho \mapsto U_1]U_2$ for the usage obtained by replacing the free occurrences of ρ in U_2 with U_1. We write $FV(U)$ for the set of free usage variables. A usage is *closed* if $FV(U) = \emptyset$.

Intuitive meaning of usages is summarized in Table 1. Additional explanation is in order. If t_o is finite, a channel of usage $\alpha^{t_o}_{t_c}.U$ must be used for the action α, while if t_o is ∞, the channel need not be used. If t_c is finite, whenever the action α is performed on a channel of usage $\alpha^{t_o}_{t_c}.U$, the action will eventually succeed. If t_c is ∞, there is no such guarantee. A channel of usage $\alpha^{t_o}_{t_c}.U$ must be used according to U

only when it has been used for the action α and the action succeeds. For example, a channel of usage $I_0^\infty.O_\infty^0$ can be used for input (but need not be used), and if it has been used for input and the input has succeeded, it *must* be used for output. Usage $\uparrow^{(t_1,t_2)}U$ is used for defining the operation $\uparrow\tau$ explained in the previous subsection. Note that obligation levels guarded by I and O are not increased. For example, $\uparrow^{(1,2)}(I_0^0.O_\infty^0 \,|\, O_0^3 \,|\, O_3^0)$ is the same as $I_0^1.O_\infty^0 \,|\, O_0^3 \,|\, O_3^2$. Choice $U_1 \,\&\, U_2$ and recursive usages $\mu\rho.U$ are only required to make a type inference complete, so that they can be skipped at first reading. Usage $\mu\rho.O_0^\infty.\rho$ describes a channel that can be used for output infinitely often.[4]

Notation 3.2: We give a higher precedence to prefixes ($\alpha_{t_c}^{t_o}$ and $*$) than to $|$. So, $I_{t_c}^{t_o}.U_1 \,|\, U_2$ means $(I_{t_c}^{t_o}.U_1) \,|\, U_2$, not $I_{t_c}^{t_o}.(U_1 \,|\, U_2)$. We write $\overline{\alpha}$ for the co-action of α ($\overline{I} = O$ and $\overline{O} = I$).

Example 3.3: Linear channels (channels that are used once for input and once for output) [17] are given a usage of the form $I_{t_2}^{t_1} \,|\, O_{t_4}^{t_3}$. For example, the channel c in Example 2.9 is given a usage $I_1^1.\,|\,O_1^1$. The usage of channels used for client-server connection (like *succ* in Example 2.6) is expressed as $*I_\infty^n \,|\, *O_n^\infty$ for some $n \in \mathbf{Nat}$. The part $*I_\infty^n$ means that a server must wait for requests forever, and the part $*O_n^\infty$ means that clients can send an infinite number of requests, and that it is guaranteed that the requests are received by a server.

Example 3.4: The usage of a lock channel (i.e., a channel used as a lock: recall Example 2.7) is expressed by $O_\infty^n \,|\, *I_n^\infty.O_\infty^n$ for some $n \in \mathbf{Nat}$. The part O_∞^n says that a value must be first put into the channel (to initialize the lock), and the part $I_n^\infty.O_\infty^n$ says that the lock can be eventually acquired, and after the lock has been acquired, then the lock must be released. The natural number n can be used to control in which order locks are acquired. Suppose that lock channels x and y have usages $*I_{n_x}^\infty.O_\infty^{n_x}$ and $*I_{n_y}^\infty.O_\infty^{n_y}$ respectively and $n_y < n_x$ holds. Then, the process $x(\,).\,y(\,).\,(\overline{y}\langle\,\rangle \,|\, \overline{x}\langle\,\rangle)$, which locks x and y in this order, is allowed, but the process $y(\,).\,x(\,).\,(\overline{x}\langle\,\rangle \,|\, \overline{y}\langle\,\rangle)$ is not allowed. Note that the latter process tries to exercise the capability of level n_x to lock x before fulfilling the obligation of level n_y to release the lock y.

Example 3.5: The usage of a channel implementing a shared variable (recall Example 2.8) is expressed as $O_\infty^0 \,|\, *I_0^\infty O_\infty^0$. It is a special case of the usage of lock channels. The capability level of an input action being 0 captures the fact that read/write operations always succeed, and the obligation level of an output action being 0 captures the fact that a value must be immediately written back when a value is extracted from the channel.

3.2.2 Semantics

The formal meaning of a usage is determined by its obligation/capability levels of a usage, which represent what capabilities/obligations currently exist, and its reduction, which express how the usage changes during reduction of processes.

We first define capability/obligation levels of a usage.

Definition 3.6 [capabilities]: The *input and output capability levels* of usage U, written $cap_I(U)$ and $cap_O(U)$, are defined by:

$$
\begin{array}{l}
cap_\alpha(\mathbf{0}) = cap_\alpha(\overline{\alpha}_{t_c}^{t_o}.U) = cap_\alpha(\rho) = \infty \\
cap_\alpha(\alpha_{t_c}^{t_o}.U) = t_c \\
cap_\alpha(*U) = cap_\alpha(\uparrow^{(t_1,t_2)}U) = cap_\alpha(\mu\rho.U) = cap_\alpha(U) \\
cap_\alpha(U_1 \,|\, U_2) = cap_\alpha(U_1 \,\&\, U_2) = \mathbf{min}(cap_\alpha(U_1), cap_\alpha(U_2))
\end{array}
$$

[4] A careful reader may think that $*U$ can be represented by $\mu\rho.(\rho \,|\, U)$. The formal semantics of $*U$ and $\mu\rho.(\rho \,|\, U)$, given later, are different in a subtle way.

Definition 3.7 [obligations]: The *input and output obligation levels* of a closed usage U, written $ob_I(U)$ and $ob_O(U)$, are defined by:

$$
\begin{array}{l}
ob_\alpha(U) = ob^\emptyset_\alpha(U) \\
ob^F_\alpha(\mathbf{0}) = ob^F_\alpha(\overline{\alpha}^{t_o}_{t_c}.U) = \infty \\
ob^F_\alpha(\rho) = F(\rho) \\
ob^F_\alpha(\alpha^{t_o}_{t_c}.U) = t_o \\
ob^F_\alpha(U_1 \,|\, U_2) = \mathbf{min}(ob^F_\alpha(U_1), ob^F_\alpha(U_2)) \\
ob^F_\alpha(\uparrow^{(t_I,t_O)}U) = \mathbf{max}(t_\alpha, ob^F_\alpha(U)) \\
ob^F_\alpha(U_1 \,\&\, U_2) = \mathbf{max}(ob^F_\alpha(U_1), ob^F_\alpha(U_2)) \\
ob^F_\alpha(*U) = ob^F_\alpha(U) \\
ob^F_\alpha(\mu\rho.U) = \mathbf{lfp}(\lambda x.ob^{F[\rho\mapsto x]}_\alpha(U))
\end{array}
$$

Here, **lfp** denotes the least fixed-point operator. We write $ob(U)$ for $\mathbf{min}(ob_I(U), ob_O(U))$.

For example, $ob_O(\mu\rho.(\rho \,|\, O^1_2) = \mathbf{lfp}(\lambda x.\mathbf{min}(x,1)) = 0$. On the other hand, $ob_O(*O^1_2) = 1$. (So, $*U$ is not the same as $\mu\rho.(\rho \,|\, U)$.)

The usage of a channel describes how the channel should be used afterwards, so the usage changes during reduction of processes. For example, if x has usage $I^\infty_\infty.O^0_0 \,|\, O^\infty_\infty.I^0_0$, after a communication on x occurs, x should be used according to $O^0_0 \,|\, I^0_0$. This change of usage is expressed by the usage reduction relation defined below. Intuitively, $U \longrightarrow U'$ means that if a channel of usage U has been used for a communication, then it should be used according to U' afterwards. As in the definition of the process reduction relation, we use a structural relation $\preceq$ as an auxiliary relation.

Definition 3.8: $\preceq$ is the least reflexive and transitive relation satisfying the following rules:

$$U_1 \,|\, U_2 \preceq U_2 \,|\, U_1 \qquad \text{(UP-Commut)}$$

$$(U_1 \,|\, U_2) \,|\, U_3 \preceq U_1 \,|\, (U_2 \,|\, U_3) \qquad \text{(UP-Assoc)}$$

$$\frac{U_1 \preceq U'_1 \qquad U_2 \preceq U'_2}{U_1 \,|\, U_2 \preceq U'_1 \,|\, U'_2} \qquad \text{(UP-CongP)}$$

$$*U \preceq *U \,|\, U \qquad \text{(UP-Rep)}$$

$$\uparrow^{(t_O,t_C)}\alpha^{t_1}_{t_2}.U \preceq \alpha^{\mathbf{max}(t_1,t_\alpha)}_{t_2}.U \qquad \text{(UP-}\uparrow\text{)}$$

$$\uparrow^{(t_1,t_2)}(U_1 \,|\, U_2) \preceq (\uparrow^{(t_1,t_2)}U_1) \,|\, (\uparrow^{(t_1,t_2)}U_2) \qquad \text{(UP-Dist)}$$

$$U_1 \,\&\, U_2 \preceq U_i \ (i \in \{1,2\}) \qquad \text{(UP-Or)}$$

$$\mu\rho.U \preceq [\rho \mapsto \mu\rho.U]U \qquad \text{(UP-Rec)}$$

$$\frac{U \preceq U'}{\uparrow^{(t_1,t_2)}U \preceq \uparrow^{(t_1,t_2)}U'} \qquad \text{(UP-Cong}\uparrow\text{)}$$

Definition 3.9 [usage reduction]: A binary relation $\longrightarrow$ on usages is the least relation closed under the following rules:

$$I^{t_o}_{t_c}.U_1 \,|\, O^{t'_o}_{t'_c}.U_2 \longrightarrow U_1 \,|\, U_2$$

$$\frac{U_1 \longrightarrow U'_1}{U_1 \,|\, U_2 \longrightarrow U'_1 \,|\, U_2}$$

$$\frac{U_1 \preceq U'_1 \qquad U'_1 \longrightarrow U'_2 \qquad U'_2 \preceq U_2}{U_1 \longrightarrow U_2}$$

3.2.3 Relations and operations on usages

The relation $rel(U)$ explained in Section 3.1 is formally defined below. It ensures that whenever there is a capability of level n to successfully perform some action, there exists an obligation of the same or lower level to do its co-action.

Definition 3.10 [reliability]: A usage U is *reliable*, written $rel(U)$, if, for any U' such that $U \longrightarrow^* U'$, $ob_\alpha(U') \leq cap_{\overline{\alpha}}(U')$ holds for each $\alpha \in \{I, O\}$.

For example, $rel(O^1_\infty \,|\, *I^\infty_1.O^1_\infty)$ holds but $rel(O^1_\infty \,|\, *I^\infty_1.O^2_\infty)$ does not. The latter usage is reduced to $O^2_\infty \,|\, *I^\infty_1.O^2_\infty$, of which the input capability level is 1 but the output obligation level is 2.

The subusage relation $U_1 \leq U_2$ defined below means that U_1 expresses more liberal usage of channels than U_2, so that a channel of usage U_1 may be used as that of usage U_2. For example, $U_1 \,\&\, U_2 \leq U_1$ and $I^2_2 \leq I^1_3$ hold. (The latter comes from the intuition that an obligation can be replaced by a stronger one, while a capability can be replaced by a weaker one.) We define the subusage relation co-inductively, by using the idea of process simulation relations.

Definition 3.11 [subusage]: The *subusage relation* $\leq$ on closed usages is the largest binary relation on usages such that the following conditions hold whenever $U_1 \leq U_2$.

- $[\rho \mapsto U_1]U \leq [\rho \mapsto U_2]U$ for any usage U such that $FV(U) = \{\rho\}$.
- If $U_2 \longrightarrow U'_2$, then there exists U'_1 such that $U_1 \longrightarrow U'_1$ and $U'_1 \leq U'_2$.
- If $rel(U_1)$ holds, then $rel(U_2)$ also holds.
- $ob(U_1) \geq ob(U_2)$.

We list some useful laws about the subusage relation.

Lemma 3.12:
1. If $U \preceq U'$, then $U \leq U'$.
2. If $ob(U) = \infty$, then $U \leq \mathbf{0}$.
3. If $t'_o \leq t_o$ and $t_c \leq t'_c$, then $\alpha^{t_o}_{t_c}.U \leq \alpha^{t'_o}_{t'_c}.U$.
4. If $U_1 \leq [\rho \mapsto U_1]U$, then $U_1 \leq \mu\rho.U$.

$\uparrow U$ defined below is the usage obtained by increasing the input and obligation levels of U by one. As explained in Section 3.1, it will be used in the typing rule for output processes.

Definition 3.13: The operation $\uparrow$ on usages is defined by: $\uparrow U = \uparrow^{(t_1+1, t_2+1)}(U)$ where $t_1 = ob_I(U)$ and $t_2 = ob_O(U)$.

For example, $\uparrow(O^1_0 \,|\, O^0_1) = \uparrow^{(\infty,1)}(O^1_0 \,|\, O^0_1)$, which is equivalent to $O^1_0 \,|\, O^1_1$.

3.3 Types

Definition 3.14 [types]: The set of *types* is given by:

$$\tau \text{ (types)} ::= \mathbf{bool}^l \mid \mathbf{unit} \mid \xi/U$$
$$\xi \text{ (core channel types)} ::= \langle \tau_1, \ldots, \tau_n \rangle^l$$

Type $\mathbf{bool}^l$ is the type of booleans whose secrecy level is l. Type **unit** is the type of the unit value $\star$. A channel type $\langle \tilde{\tau} \rangle^l / U$ describes a channel that have secrecy level l and should be used according to U for communicating tuples of values of types $\tilde{\tau}$.

Throughout this paper, we assume that channel types always satisfy the following well-formedness conditions.

Definition 3.15: A channel type $\langle \tilde{\tau} \rangle^l / U$ is *well-formed* if it satisfies the following two conditions:

- If $l = \mathbf{H}$, then all the secrecy annotations in $\tilde{\tau}$ are $\mathbf{H}$.
- If $l = \mathbf{L}$, then all the capability level annotations in U are ∞.

It is possible to remove the above conditions, but those conditions make the typing rules and the proof of type soundness simpler. The first condition ensures that only secret data can be transmitted through a secret channel. This is a reasonable assumption, since if a channel is secret, a low-level observer cannot extract data from the channel. The second condition is also reasonable, since, for the purpose of information flow analysis, analyzing whether communications succeed is important only for secret channels.

We extend relations and operations on usages to those on types.

Definition 3.16 [subtyping]: A subtyping relation $\leq$ is the least reflexive relation closed under the following rule:

$$\frac{U \leq U'}{\xi/U \leq \xi/U'} \qquad \text{(SubT-Chan)}$$

For the sake of simplicity, we do not consider subtyping based on secrecy levels (e.g. $\mathbf{bool}^{\mathbf{L}} \leq \mathbf{bool}^{\mathbf{H}}$) and input/output modes [23]. Extension to allow such subtyping is discussed in Section 6.

Definition 3.17: The *obligation level* of type τ, written $ob(\tau)$, is defined by: $ob(\mathbf{unit}) = ob(\mathbf{bool}^l) = \infty$ and $ob(\xi/U) = ob(U)$.

Definition 3.18: Unary operations $*$ and $\uparrow$ on types is defined by: $\uparrow\mathbf{unit} = \mathbf{unit}$, $\uparrow\mathbf{bool}^l = \mathbf{bool}^l$, $\uparrow(\xi/U) = \xi/\uparrow U$, $*\mathbf{unit} = \mathbf{unit}$, $*\mathbf{bool}^l = \mathbf{bool}^l$, and $*(\xi/U) = \xi/*U$.

Definition 3.19: A (partial) binary operation $\mid$ on types is defined by: $\mathbf{unit} \mid \mathbf{unit} = \mathbf{unit}$, $\mathbf{bool}^l \mid \mathbf{bool}^l = \mathbf{bool}^l$, and $(\xi/U_1) \mid (\xi/U_2) = \xi/(U_1 \mid U_2)$. $\tau_1 \mid \tau_2$ is undefined if it does not match any of the above rules.

3.4 Type Environment

A type environment is a mapping from a finite set of variables to types. We use metavariables Γ and Δ for type environments. We write $\emptyset$ for the type environment whose domain is empty. When $x \notin dom(\Gamma)$, we write $\Gamma, x : \tau$ for the type environment Γ' such that $dom(\Gamma') = dom(\Gamma) \cup \{x\}$, $\Gamma'(x) = \tau$, and $\Gamma'(y) = \Gamma(y)$ for $y \in dom(\Gamma)$.

$$\emptyset \vdash_l \mathbf{0} \qquad \text{(T-Zero)}$$

$$\frac{\Gamma \vdash_l P}{*\Gamma \vdash_l *P} \qquad \text{(T-Rep)}$$

$$\frac{\Gamma, x:\langle\tilde{\tau}\rangle^{l_1}/U \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad \tilde{\tau'} \leq \uparrow\tilde{\tau} \qquad t_c + 1 \leq ob(\Gamma \,|\, \tilde{v}:\tilde{\tau'}) \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\Gamma \,|\, \tilde{v}:\tilde{\tau'} \,|\, x:\langle\tilde{\tau}\rangle^{l_1}/O^0_{t_c}.U \vdash_l \overline{x}\langle\tilde{v}\rangle.\, P} \qquad \text{(T-Out)}$$

$$\frac{\Gamma, x:\xi/U \vdash_l P \qquad rel(U)}{\Gamma \vdash_l (\nu x:\xi)\, P} \qquad \text{(T-New)}$$

$$\frac{\Gamma \vdash_l P \qquad \Gamma \vdash_l Q}{\Gamma \,|\, v:\mathbf{bool}^l \vdash_l \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q} \qquad \text{(T-If)}$$

$$\frac{\Gamma, x:\langle\tilde{\tau}\rangle^{l_1}/U, \tilde{y}:\tilde{\tau} \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \qquad t_c + 1 \leq ob(\Gamma) \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\Gamma, x:\langle\tilde{\tau}\rangle^{l_1}/I^0_{t_c}.U \vdash_l x(\tilde{y}).\, P} \qquad \text{(T-In)}$$

$$\frac{\Gamma' \vdash_{l'} P \qquad \Gamma \leq \Gamma' \qquad l \sqsubseteq l'}{\Gamma \vdash_l P} \qquad \text{(T-Weak)}$$

$$\frac{\Gamma_1 \vdash_l P_1 \qquad \Gamma_2 \vdash_l P_2}{\Gamma_1 \,|\, \Gamma_2 \vdash_l P_1 \,|\, P_2} \qquad \text{(T-Par)}$$

Figure 2: Typing Rules

A type environment Γ is extended to a mapping from a finite set of variables and constants to types, by $\Gamma(\mathit{true}^l) = \Gamma(\mathit{false}^l) = \mathbf{bool}^l$ and $\Gamma(\star) = \mathbf{unit}$. When v is a constant, $\Gamma, v:\tau$ is defined to be Γ only when $\Gamma(v) = \tau$. We abbreviate $\emptyset, v_1:\tau_1, \ldots, v_n:\tau_n$ to $v_1:\tau_1, \ldots, v_n:\tau_n$.

The operations and relations on types are pointwise extended to those on type environments below.

The subtyping relation is extended to a relation on type environments. $\Gamma_1 \leq \Gamma_2$ means that Γ_1 represents more liberal usage of free variables than Γ_2.

Definition 3.20: A binary relation $\leq$ on type environments is defined by: $\Gamma_1 \leq \Gamma_2$ if and only if (i) $dom(\Gamma_1) \supseteq dom(\Gamma_2)$, (ii) $\Gamma_1(x) \leq \Gamma_2(x)$ for each $x \in dom(\Gamma_2)$, and (iii) $ob(\Gamma_1(x)) = \infty$ for each $x \in dom(\Gamma_1) \backslash dom(\Gamma_2)$.

Definition 3.21: Suppose $dom(\Gamma) = \{x_1, \ldots, x_n\}$. We write $ob(\Gamma)$ for $\mathbf{min}(ob(\Gamma(x_1)), \ldots, ob(\Gamma(x_n)))$.

Definition 3.22: The operations $|$ and $*$ on type environments are defined by:

$$(\Gamma_1 \,|\, \Gamma_2)(x) = \begin{cases} \Gamma_1(x) \,|\, \Gamma_2(x) & \text{if } x \in dom(\Gamma_1) \cap dom(\Gamma_2) \\ \Gamma_1(x) & \text{if } x \in dom(\Gamma_1) \backslash dom(\Gamma_2) \\ \Gamma_2(x) & \text{if } x \in dom(\Gamma_2) \backslash dom(\Gamma_1) \end{cases}$$

$$(*\Gamma)(x) = *(\Gamma(x))$$

3.5 Typing Rules

A type judgment is of the form $\Gamma \vdash_l P$, which should be read "P is well typed under Γ and has secrecy level l." It means that P uses free variables as specified by Γ, and the secrecy level of information about its behavior is l. The typing rules for deriving valid type judgments are given in Figure 2. In the rules, "$\Gamma, \tilde{x}:\tilde{\tau}$" and "$\Gamma \,|\, \tilde{x}:\tilde{\tau}$" are abbreviations for "$\Gamma, x_1:\tau_1, \ldots, x_n:\tau_n$" and "$\Gamma \,|\, x_1:\tau_1 \,|\, \cdots \,|\, x_n:\tau_n$" respectively. $\tilde{\tau'} \leq \tilde{\tau}$ is an abbreviation for $\tau'_1 \leq \tau_1, \ldots, \tau'_n \leq \tau_n$. We explain some rules below. Rules T-In, T-Par, and T-New have already been explained in Section 3.1.

T-Out: The condition $t_c + 1 \leq ob(\Gamma \,|\, \tilde{v}:\tilde{\tau'})$ enforce Rule A given in Section 3.1. It ensures that the level of obligations held by the output process is greater than the level of the capability of the output on x being used. Note that the types of $\tilde{v}$ in the type environment of the conclusion must be subtypes of $\uparrow\tilde{\tau}$ rather than $\tilde{\tau}$: As explained in Section 3.1, it prevents infinite delegations of obligations. The obligation level of the output on x is 0, since it is fulfilled immediately.

T-If: In combination with T-Weak, the rule ensures that the secrecy level of the boolean is less than or equal to the secrecy levels of the then-part and the else-part. Note that the type environment

$\Gamma \mid v : \mathbf{bool}^{l}$ in the conclusion implicitly assumes that $\Gamma \mid v : \mathbf{bool}^{l}$ is well defined; so, the process cannot be typed if $v = \star$. The rule can actually be replaced by the following, less restrictive rule:

$$\frac{\Gamma \vdash_l P \qquad \Gamma \vdash_l Q \qquad l' \sqsubseteq l}{\Gamma \mid v : \mathbf{bool}^{l'} \vdash_l \mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q}$$

This rule avoids propagation of the secrecy level of the boolean to the if-expression. If we extend the subtyping relation based on the secrecy level (as discussed in Section 6), the two rules become equivalent.

T-Weak: This rule allows the type environment to be replaced by a type environment expressing more liberal uses of channels, and the secrecy level of the process to be replaced by a lower one. For example, from $x : \xi/O^0_\infty \vdash_{\mathbf{H}} P$, we can obtain $x : \xi/O^\infty_0 \vdash_{\mathbf{L}} P$.

3.6 Examples

Example 3.23: The process $(\nu x : \langle\,\rangle^{\mathbf{H}})\,(x(\,).\,\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ is typed as follows:

$$\dfrac{\dfrac{\dfrac{y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} \overline{y}\langle\,\rangle}{x : \langle\,\rangle^{\mathbf{H}}/I^0_0, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} x(\,).\,\overline{y}\langle\,\rangle}\ \text{T-In} \quad \dfrac{x : \langle\,\rangle^{\mathbf{H}}/O^0_0 \vdash_{\mathbf{H}} \overline{x}\langle\,\rangle}{x : \langle\,\rangle^{\mathbf{H}}/O^0_0 \vdash_{\mathbf{L}} \overline{x}\langle\,\rangle}\ \text{T-Weak}}{x : \langle\,\rangle^{\mathbf{H}}/(I^0_0 \mid O^0_0), y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} x(\,).\,\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle}\ \text{T-Par}}{y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} (\nu x : \langle\,\rangle^{\mathbf{H}})\,(x(\,).\,\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)}\ \text{T-New}$$

The secrecy level of y can be $\mathbf{L}$, although that of x is declared as $\mathbf{H}$.

Example 3.24: Let us consider the following process P:

$$*s(r).\,\overline{r}\langle\,\rangle \mid \overline{s}\langle x\rangle \mid x(\,).\,\overline{y}\langle\,\rangle$$

It is well typed under the following type environment:

$$s : \langle\langle\,\rangle^{\mathbf{H}}/O^0_\infty\rangle^{\mathbf{H}}/(*I^0_\infty \mid O^\infty_0), x : \langle\,\rangle^{\mathbf{H}}/(O^1_\infty \mid I^\infty_1), y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty$$

Note that the type system can infer that the input on x succeeds, so that y is assigned level $\mathbf{L}$ although the secrecy level of x is $\mathbf{H}$.

Example 3.25: Suppose that x has type $\langle\,\rangle^{\mathbf{H}}/*I^\infty_{n_x}.O^{n_x}_\infty$ and y has type $\langle\,\rangle^{\mathbf{H}}/*I^\infty_{n_y}.O^{n_y}_\infty$ with $n_y < n_x$. Then, as mentioned in Example 3.4, the process $x(\,).\,y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ is allowed which locks x and y in this order is allowed, but the process $y(\,).\,x(\,).\,(\overline{x}\langle\,\rangle \mid \overline{y}\langle\,\rangle)$ is not. In fact, $x(\,).\,y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ is typed as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{x : \langle\,\rangle^{\mathbf{H}}/O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/O^{n_y}_\infty \vdash_{\mathbf{H}} \overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle}}{x : \langle\,\rangle^{\mathbf{H}}/O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/I^0_{n_y}.O^{n_y}_\infty \vdash_{\mathbf{H}} y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)}\ \text{T-In}}{x : \langle\,\rangle^{\mathbf{H}}/O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/I^\infty_{n_y}.O^{n_y}_\infty \vdash_{\mathbf{H}} y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)}\ \text{T-Weak}}{x : \langle\,\rangle^{\mathbf{H}}/I^0_{n_x}.O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/I^\infty_{n_y}.O^{n_y}_\infty \vdash_{\mathbf{H}} x(\,).\,y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)}\ \text{T-In}}{x : \langle\,\rangle^{\mathbf{H}}/*I^\infty_{n_x}.O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/*I^\infty_{n_y}.O^{n_y}_\infty \vdash_{\mathbf{H}} x(\,).\,y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)}\ \text{T-Weak}$$

The first application of T-In is valid since $ob(x : \langle\,\rangle^{\mathbf{H}}/O^{n_x}_\infty) = n_x \geq n_y + 1$.

On the other hand,

$$x : \langle\,\rangle^{\mathbf{H}}/*I^\infty_{n_x}.O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/*I^\infty_{n_y}.O^{n_y}_\infty \vdash_{\mathbf{H}} y(\,).\,x(\,).\,(\overline{x}\langle\,\rangle \mid \overline{y}\langle\,\rangle)$$

is not derivable, since we cannot apply T-In to obtain

$$x : \langle\,\rangle^{\mathbf{H}}/I^0_{n_x}.O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/O^{n_y}_\infty \vdash_{\mathbf{H}} x(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$$

from $x : \langle\,\rangle^{\mathbf{H}}/O^{n_x}_\infty, y : \langle\,\rangle^{\mathbf{H}}/O^{n_y}_\infty \vdash_{\mathbf{H}} \overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle$. Notice that the required condition: $n_x{+}1 \leq ob(y : \langle\,\rangle^{\mathbf{H}}/O^{n_y}_\infty) = n_y$ does not hold.

Example 3.26: The process A in Example 2.9 has the secrecy level **L** under the following type environment:

$$\begin{array}{l} \mathit{secret} : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0 O^0_\infty, \mathit{public} : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/{*}I^\infty_\infty.O^\infty_\infty, \\ x : \langle\,\rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty \end{array}$$

(The bound channel c is assigned a type $\langle\,\rangle^{\mathbf{H}}/(O^1_1 \mid I^1_1)$.)

The process $C = y(\,).\,x(\,).\,(\overline{x}\langle\,\rangle \mid \overline{y}\langle\,\rangle)$ is well typed under $x : \langle\,\rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty, y : \langle\,\rangle^{\mathbf{L}}/{*}I^\infty_\infty.O^\infty_\infty$. So, the process $(\nu x)\,(\nu \mathit{secret})\,(A \mid C \mid \overline{x}\langle\,\rangle \mid \overline{\mathit{secret}}\langle b\rangle)$ is well typed where b is $\mathit{true}^{\mathbf{H}}$ or $\mathit{true}^{\mathbf{H}}$. (Here, $\overline{x}\langle\,\rangle \mid \overline{\mathit{secret}}\langle b\rangle$ initializes the lock x and the shared variable *secret*. So, we know that the concurrent execution of threads A and C in Figure 1 is safe.

On the other hand, the process $B = x(\,).\,y(\,).\,(\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle)$ is only well-typed under $x : \langle\,\rangle^{\mathbf{H}}/{*}I^\infty_\infty.O^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/{*}I^\infty_\infty.O^\infty_\infty$. So, $(\nu x)\,(\nu \mathit{secret})\,(A \mid B \mid \overline{x}\langle\,\rangle \mid \overline{\mathit{secret}}\langle b\rangle)$ is not well typed (since the whole usage of x: ${*}I^\infty_\infty.O^\infty_\infty \mid {*}I^\infty_0.O^0_\infty \mid O^0_\infty$ is not reliable), which implies that the concurrent execution of threads A and B may leak secret information.

Example 3.27: Let us consider the process given at the end of Example 2.10. Suppose that Γ_1, Γ_2 and Γ_3 are given as follows.

$$\begin{array}{l} \Gamma_1 = x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty, c : \langle\,\rangle^{\mathbf{H}}/O^1_0, c_1 : \langle\,\rangle^{\mathbf{H}}/O^1_\infty \\ \Gamma_2 = x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty, y : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty, c : \langle\,\rangle^{\mathbf{H}}/I^0_1, c_2 : \langle\,\rangle^{\mathbf{H}}/O^2_\infty \\ \Gamma_3 = c_1 : \langle\,\rangle^{\mathbf{H}}/I^0_1, c_2 : \langle\,\rangle^{\mathbf{H}}/I^2_2, w : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/{*}I^\infty_\infty.O^\infty_\infty \end{array}$$

Then, we have:

$$\begin{array}{l} \Gamma_1 \vdash_{\mathbf{H}} x(z).\,(\overline{x}\langle \mathit{true}^{\mathbf{H}} \rangle \mid \overline{c}\langle\,\rangle.\,\overline{c_1}\langle\,\rangle) \\ \Gamma_2 \vdash_{\mathbf{H}} c(y).\,x(z).\,(\overline{x}\langle z\rangle \mid y(u).\,(\overline{y}\langle z\rangle \mid \overline{c_2}\langle\,\rangle)) \\ \Gamma_3 \vdash_{\mathbf{L}} c_1(\,).\,c_2(\,).\,w(z)\,.\,\overline{w}\langle \mathit{false}^{\mathbf{L}} \rangle \end{array}$$

The whole process is well-typed under the type environment:

$$x : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty, y : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty, c : \langle\,\rangle^{\mathbf{H}}/(O^1_0 \mid I^0_1), w : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/{*}I^\infty_\infty.O^\infty_\infty$$

So, although the part `cobegin ... coend` performs synchronization on high-level channels, our type system can correctly infer that it does not affect the execution of the part `w := false`.

Example 3.28: Let us reconsider the process in Example 3.28. Subprocesses ${*}\mathit{withdraw}(\mathit{amount}, r).\,\cdots$, ${*}\mathit{deposit}(\mathit{amount}, r).\,\cdots$, and ${*}\mathit{transfer}(\mathit{amount}, r).\,\cdots$ are typed as follows. (Here, we assume that the type system is extended with integer types.)

$$\begin{array}{l} \mathit{withdraw} : \langle \mathit{int}^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}}/O^1_\infty \rangle^{\mathbf{H}}/{*}I^0_\infty, s : \langle \mathit{int}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty \\ \qquad\qquad \vdash_{\mathbf{H}} {*}\mathit{withdraw}(\mathit{amount}, r).\,\cdots \\ \mathit{deposit} : \langle \mathit{int}^{\mathbf{H}}, \langle \mathit{int}^{\mathbf{H}} \rangle^{\mathbf{H}}/O^1_\infty \rangle^{\mathbf{H}}/{*}I^0_\infty, s : \langle \mathit{int}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty \\ \qquad\qquad \vdash_{\mathbf{H}} {*}\mathit{deposit}(\mathit{amount}, r).\,\cdots \\ \mathit{transfer} : \langle \mathit{int}^{\mathbf{L}}, \langle\,\rangle^{\mathbf{L}}/O^1_\infty \rangle^{\mathbf{L}}/{*}I^0_\infty, s : \langle \mathit{int}^{\mathbf{H}} \rangle^{\mathbf{H}}/{*}I^\infty_0.O^0_\infty \\ \qquad\qquad \vdash_{\mathbf{L}} {*}\mathit{transfer}(\mathit{amount}, r).\,\cdots \end{array}$$

The key is the typing for the last sub-process. Although it performs communication on a high-level channel s, it is allowed to send a message on a low-level channel r, since our type system guarantees that the input

on s always succeeds. (Note that the capability level of an input on s is 0.) The whole process is well-typed under:

$$\begin{array}{l} \mathit{withdraw} : \langle \mathit{int}^{\mathbf{H}}, \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / *I^0_\infty, \mathit{deposit} : \langle \mathit{int}^{\mathbf{H}}, \langle \mathit{int}^{\mathbf{H}} \rangle^{\mathbf{H}} / O^1_\infty \rangle^{\mathbf{H}} / *I^0_\infty, \\ \mathit{transfer} : \langle \mathit{int}^{\mathbf{L}}, \langle\,\rangle^{\mathbf{L}} / O^1_\infty \rangle^{\mathbf{L}} / *I^0_\infty \end{array}$$

This implies that information about the current balance is not leaked through the public channel *transfer*.

4 Soundness of the Type System

In this section, we show that well-typed processes satisfy a so-called non-interference property, which says that high-level values and processes (i.e., values and processes of the secrecy level **H**) do not affect the behavior of low-level processes. The property implies that information about high-level values or processes cannot be observed by low-level processes.

Before proving the non-interference property, we show a subject reduction theorem. As in other linear type systems for process calculi [17], type environments may change during reduction. We write $\Gamma \longrightarrow \Gamma'$ when $\Gamma = \Gamma_1, x : \xi / U$ and $\Gamma' = \Gamma_1, x : \xi / U'$ with $U \longrightarrow U'$ for some Γ_1, ξ, U, and U'.

Theorem 4.1 [subject reduction]: If $\Gamma \vdash_l P$ and $P \longrightarrow Q$, then $\Delta \vdash_l Q$ holds for some Δ such that $\Gamma = \Delta$ or $\Gamma \longrightarrow \Delta$.

Proof: See Appendix C. □

In order to formally state the non-interference property, we define a process equivalence relation based on the notion of barbed congruence [28]. The idea of barbed congruence is to put two processes into various contexts and check whether they exhibit the same observational behavior. The set of observables, called *barbs*, is defined as follows.

Definition 4.2 [barbs]: The *barbs* of P, written $\mathit{Barbs}(P)$, is defined by:

$$\begin{array}{rcl} \mathit{Barbs}(P) & = & \{\overline{x} \mid P \longrightarrow^* \preceq (\nu \tilde{y})\,(\overline{x}\langle \tilde{v} \rangle . \, Q \mid R), x \notin \{\tilde{y}\}\} \\ & & \cup \{x \mid P \longrightarrow^* \preceq (\nu \tilde{y})\,(x(\tilde{z}) . \, Q \mid R), x \notin \{\tilde{y}\}\} \end{array}$$

The two processes put into the same context are compared by using the following barbed bisimulation.

Definition 4.3 [barbed bisimulation]: A binary relation $\mathcal{R}$ on processes is a *barbed bisimulation* if the following conditions hold for every $(P, Q) \in \mathcal{R}$:

- If $P \longrightarrow P'$, then there exists Q' such that $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$,
- If $Q \longrightarrow Q'$, then there exists P' such that $P \longrightarrow^* P'$ and $(P', Q') \in \mathcal{R}$, and
- $\mathit{Barbs}(P) = \mathit{Barbs}(Q)$.

P and Q are barbed bisimilar, written $P \stackrel{\bullet}{\approx} Q$, if $(P, Q) \in \mathcal{R}$ holds for some barbed bisimulation.

The definition of contexts is given as follows.

Definition 4.4 [context]: A *context* is a term obtained from a process by replacing a sub-process with $[\,]$. We write $C[P]$ for the process obtained by replacing $[\,]$ in C with P. A context C is a (Γ, l)-(Δ, l')-context if $\Delta \vdash_{l'} C$ is derivable from $\Gamma \vdash_l [\,]$.

We introduce some terminology about type environments. A type environment Γ is *low-level* if all the secrecy level annotations appearing in Γ is $\mathbf{L}$. A type environment Γ is *ground* if $\Gamma(x)$ is a channel type for any $x \in dom(\Gamma)$. Γ is *closed* if for any $x \in dom(\Gamma)$, $\Gamma(x)$ is a channel type of the form ξ/U and $rel(U)$ holds.

Now we can define the barbed congruence. Basically, two processes P and Q are barbed congruent if $C[P]$ and $C[Q]$ are barbed bisimilar for an arbitrary context C. Here, since we are dealing with well-typed processes, we consider only "well-typed" contexts.

Definition 4.5 [barbed congruence]: P and Q are barbed (Γ, l)-congruent, written $P \approx_{\Gamma,l} Q$, if (i) $\Gamma \vdash_l P$, (ii) $\Gamma \vdash_l Q$, and (iii) for any ground Δ and secrecy level l', $C[P] \stackrel{\bullet}{\approx} C[Q]$ holds for any (Γ, l)-(Δ, l')-context C.

We can now state the non-interference property as the following theorems. The first one says that the difference between high-level *values* is not observable to low-level processes, and the second one says that the difference between high-level *processes* is not observable to low-level processes. The second one is required to prevent leakage of information about complex data structures, which are represented as processes in the π-calculus [28].

Theorem 4.6 [non-interference (1)]: Suppose that Δ is a low-level type environment. If $\Delta \vdash_l [x \mapsto \mathit{true}^{\mathbf{H}}]P$ holds, then $[x \mapsto \mathit{true}^{\mathbf{H}}]P \approx_{\Delta,l} [x \mapsto \mathit{false}^{\mathbf{H}}]P$.

Theorem 4.7 [non-interference (2)]: Suppose that Δ is a low-level type environment, and that C is a $(\Gamma, \mathbf{H}) - (\Delta, l)$-context. If $\Gamma \vdash_{\mathbf{H}} P_i$ holds for $i = 1, 2$, then $C[P_1] \approx_{\Delta,l} C[P_2]$.

The central idea of the proofs of the above theorems is that if $\Delta \vdash_l P$ holds for a low-level, closed typed environment Δ, then P and the process obtained from P by eliminating all the high-level values and processes are equivalent. In Theorem 4.7, $C[P_1]$ and $C[P_2]$ are erased to the same process (up to a structural relation), so that $C[P_1] \approx \mathbf{Er}(C[P_1]) \approx \mathbf{Er}(C[P_2]) \approx C[P_2]$ holds (where $\mathbf{Er}$ is the function that eliminates all the high-level values and processes).

Below we define $\mathbf{Er}$ formally and then prove Theorem 4.7.

We write $\mathbf{High}(\tau)$ if τ is $\mathbf{unit}$, $\mathbf{bool}^{\mathbf{H}}$, or a channel type of the form $\langle \tilde{\tau} \rangle^{\mathbf{H}}/U$.

Definition 4.8: $\mathbf{Er}_\Gamma(P)$ is defined by:

$$\mathbf{ErV}_\Gamma(v) = \begin{cases} \star & \text{if } \mathbf{High}(\Gamma(v)) \\ v & \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(\mathbf{0}) = \mathbf{0}$$

$$\mathbf{Er}_\Gamma(\overline{x}\langle \tilde{v} \rangle . P) = \begin{cases} \overline{x}\langle \tilde{v'} \rangle . \mathbf{Er}_\Gamma(P) \\ \qquad \text{if } \Gamma(x) = \langle \tilde{\tau} \rangle^{\mathbf{L}}/U \text{ and } v'_i = \mathbf{ErV}_\Gamma(v_i) \\ \mathbf{Er}_\Gamma(P) \qquad \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(x(\tilde{y}) . P) = \begin{cases} x(\tilde{y}) . \mathbf{Er}_{\Gamma, \tilde{y} : \tilde{\tau}}(P) & \text{if } \Gamma(x) = \langle \tilde{\tau} \rangle^{\mathbf{L}}/U \\ \mathbf{Er}_{\Gamma, \tilde{y} : \tilde{\tau}}(P) & \text{if } \Gamma(x) = \langle \tilde{\tau} \rangle^{\mathbf{H}}/U \\ \mathbf{Er}_{\Gamma, \tilde{y} : \tilde{\mathbf{unit}}}(P) & \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(P \mid Q) = \mathbf{Er}_\Gamma(P) \mid \mathbf{Er}_\Gamma(Q)$$

$$\mathbf{Er}_\Gamma(*P) = *\mathbf{Er}_\Gamma(P)$$

$$\mathbf{Er}_\Gamma((\nu x : \xi)\, P) = \begin{cases} (\nu x : \xi)\, \mathbf{Er}_{\Gamma, x : \xi/\mathbf{0}}(P) & \text{if } \xi = \langle \tilde{\tau} \rangle^{\mathbf{L}} \\ \mathbf{Er}_{\Gamma, x : \xi/\mathbf{0}}(P) & \text{otherwise} \end{cases}$$

$$\mathbf{Er}_\Gamma(\mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q) = \begin{cases} \mathbf{if}\ v\ \mathbf{then}\ \mathbf{Er}_\Gamma(P)\ \mathbf{else}\ \mathbf{Er}_\Gamma(Q) \\ \qquad \text{if } \Gamma(v) = \mathbf{bool}^{\mathbf{L}} \\ \mathbf{0} \quad \text{otherwise} \end{cases}$$

The following lemma shows that all high-level processes are erased to $\mathbf{0}$. (Recall that $P \equiv Q$ means $P \preceq Q$ and $Q \preceq P$.)

Lemma 4.9: If $\Gamma \vdash_{\mathbf{H}} P$, then $\mathbf{Er}_\Gamma(P) \equiv \mathbf{0}$.

Proof: Straightforward induction on derivation of $\Gamma \vdash_{\mathbf{H}} P$. □

The following theorem says that the erasure function preserves barbed bisimilarity.

Theorem 4.10: Suppose that Γ is a low-level, closed type environment. If $\Gamma \vdash_l P$, then $P \stackrel{\bullet}{\approx} \mathbf{Er}_\Gamma(P)$.

We show only a proof sketch. The full proof is given in Appendix D.

Proof sketch: Let $\mathcal{R} = \{(P, Q) \mid \Gamma \vdash_l P,\ Q \preceq \mathbf{Er}_\Gamma(P),\ \Gamma \text{ is low-level and closed}\}$. We show that $\mathcal{R}$ is a barbed bisimulation. Assuming $(P, Q) \in \mathcal{R}$, we check the three conditions of Definition 4.3. The first and second conditions follow from the following observation.

- Any reduction of P on a low-level channel is matched by the corresponding reduction of $\mathbf{Er}_\Gamma(P)$,
- Any reduction of P on a high-level channel is matched by the skip $\mathbf{Er}_\Gamma(P) \longrightarrow^* \mathbf{Er}_\Gamma(P)$,
- Any reduction of $\mathbf{Er}_\Gamma(P)$ is matched by the corresponding reduction of P, preceded by some reductions of P on high-level channels with finite capability levels. (Here, we use the fact that any input or output action of a finite capability level succeeds.)

The third condition can be checked in a similar manner. □ □

For example, let us consider a process $P = (\nu x : \langle\,\rangle^{\mathbf{H}})\,(x(\,).\,\overline{y}\langle\,\rangle \mid \overline{x}\langle\,\rangle.\,y(\,))$. $\mathbf{Er}_{y:\langle\,\rangle^{\mathbf{L}}/(O_\infty^\infty \mid I_\infty^\infty)}(P) = \overline{y}\langle\,\rangle \mid y(\,)$. The reduction of the erased process: $\overline{y}\langle\,\rangle \mid y(\,) \longrightarrow \mathbf{0}$ is simulated by the following two reduction steps:

$$P \longrightarrow (\nu x : \langle\,\rangle^{\mathbf{H}})\,(\overline{y}\langle\,\rangle \mid y(\,)) \longrightarrow (\nu x : \langle\,\rangle^{\mathbf{H}})\,\mathbf{0}$$

We write $\mathbf{low}(\Gamma)$ for the type environment obtained from Γ by replacing all the secrecy annotations with $\mathbf{L}$ and all the capability level annotations with ∞. We also write $\mathbf{low}(C)$ for the context obtained from C by replacing all the secrecy annotations with $\mathbf{L}$ and all the capability level annotations with ∞. For example, $\mathbf{low}((\nu x : \langle\langle\,\rangle^{\mathbf{H}}/O_1^0\rangle^{\mathbf{L}})\,(\mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ \overline{x}\langle y\rangle\ \mathbf{else}\ [\,])) = (\nu x : \langle\langle\,\rangle^{\mathbf{L}}/O_\infty^0\rangle^{\mathbf{L}})\,(\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ \overline{x}\langle y\rangle\ \mathbf{else}\ [\,])$.

Lemma 4.11: Suppose that Γ is a low-level type environment. If C is a (Γ, l)-(Δ, l')-context, then $\mathbf{low}(C)$ is a (Γ, l)-$(\mathbf{low}(\Delta), \mathbf{L})$-context.

The erasure function is extended to that on contexts by $\mathbf{Er}_\Gamma([\,]) = [\,]$. The following lemma follows by straightforward induction on the structure of C.

Lemma 4.12: Suppose that $\Gamma \vdash_l P$ and that C is a $(\Gamma, l) - (\Delta, l')$ context. Then, $\mathbf{Er}_\Delta(C[P]) = \mathbf{Er}_\Delta(C)[\mathbf{Er}_\Gamma(P)]$.

Now we can prove the non-interference theorem.

Proof of Theorem 4.7: Suppose that Δ' is ground. Then $\mathbf{low}(\Delta')$ is a low-level, closed type environment (since all the capability level annotations have been replaced by ∞). Let C_1 be a (Δ, l)-(Δ', l')-context. Let C_1' be $\mathbf{low}(C_1)$. By Lemma 4.11, C_1' is a (Δ, l)-$(\mathbf{low}(\Delta'), \mathbf{L})$-context. By Lemma 4.9 and Theorem 4.10, $C_1'[C[P_1]] \stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta')}(C_1'[C[P_1]]) = \mathbf{Er}_{\mathbf{low}(\Delta')}(C_1'[C])[\mathbf{Er}_\Gamma(P_1)] \stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta')}(C_1'[C])[\mathbf{0}] \stackrel{\bullet}{\approx} \mathbf{Er}_{\mathbf{low}(\Delta')}(C_1'[C])[\mathbf{Er}_\Gamma(P_2)] = \mathbf{Er}_{\mathbf{low}(\Delta')}(C_1'[C[P_2]]) \stackrel{\bullet}{\approx} C_1'[C[P_2]]$. Since $\stackrel{\bullet}{\approx}$ does not depend on type annotation, $C_1[C[P_1]] \stackrel{\bullet}{\approx} C_1[C[P_2]]$ holds. □ □

5 Type Inference Algorithm

This section describes a type inference algorithm. By the soundness of the type system, the type inference algorithm can be used to check that processes do not leak secret information. An input of the algorithm is a triple (Γ, P, l), where types and secrecy levels appearing in the triple may contain variables (to represent unknown types, secrecy levels, etc.), and all the usages in the triple must be variables. The algorithm answers whether there exists a substitution θ such that $\theta\Gamma \vdash_{\theta l} \theta P$. (The algorithm can be modified to also output a set of constraints that contain all the correct substitutions.)

The algorithm is sound and complete (in the sense that it always terminates and gives a correct yes/no-answer) with respect to a slightly less expressive variant of the type system, which is obtained by replacing T-Out and T-In with the following rules:

$$\frac{\begin{array}{c}\Gamma, x : \langle \tilde{\tau} \rangle^{l_1}/U \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \\ t_c = \infty \Rightarrow l_1 \sqsubseteq l_2\end{array}}{\uparrow^{(t_c+1,t_c+1)}(\Gamma \mid \tilde{v} : \uparrow\tilde{\tau}) \mid x : \langle \tilde{\tau} \rangle^{l_1}/O^0_{t_c}.U \vdash_l \overline{x}\langle \tilde{v} \rangle.\, P} \qquad \text{(T-Out')}$$

$$\frac{\begin{array}{c}\Gamma, x : \langle \tilde{\tau} \rangle^{l_1}/U, \tilde{y} : \tilde{\tau} \vdash_{l_2} P \qquad l \sqsubseteq l_1, l_2 \\ t_c = \infty \Rightarrow l_1 \sqsubseteq l_2\end{array}}{\uparrow^{(t_c+1,t_c+1)}\Gamma, x : \langle \tilde{\tau} \rangle^{l_1}/I^0_{t_c}.U \vdash_l x(\tilde{y}).\, P} \qquad \text{(T-In')}$$

Here, $\uparrow^{(t_1,t_2)}\Gamma$ is the pointwise extension of the operation $\uparrow^{(t_1,t_2)}$ on usages.

The above rules can be derived from T-Out, T-In, and T-Weak, so that the resulting type system is also sound. For example, T-Out' can be derived as follows:

$$\dfrac{\dfrac{\Gamma, x : \langle \tilde{\tau} \rangle^{l_1}/U \vdash_{l_2} P}{\uparrow^{(t_c+1,t_c+1)}\Gamma, x : \langle \tilde{\tau} \rangle^{l_1}/U \vdash_{l_2} P}\ \text{T-Weak} \qquad \uparrow^{(t_c+1,t_c+1)}\uparrow\tilde{\tau} \leq \uparrow\tilde{\tau}}{\uparrow^{(t_c+1,t_c+1)}\Gamma \mid \tilde{v} : \uparrow^{(t_c+1,t_c+1)}\uparrow\tilde{\tau} \mid x : \langle \tilde{\tau} \rangle^{l_1}/O^0_{t_c}.U \vdash_l \overline{x}\langle \tilde{v} \rangle.\, P}\ \text{T-In}$$

Note that the above variant is still expressive enough to deal with the examples given in Section 2. An example that is accepted by the type system of Section 3 but rejected by the above variant is:

$$x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} (\nu z)\,(x(\,).\,(z(\,).\,\overline{y}\langle\,\rangle \mid P))$$

where $P = \mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ \overline{x}\langle\,\rangle\ \mathbf{else}\ \overline{x}\langle\,\rangle$. This can be derived in the type system of Section 3 as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\cdots}{x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty, z : \langle\,\rangle^{\mathbf{H}}/(I^0_0 \mid O^0_0) \vdash_{\mathbf{L}} z(\,).\,\overline{y}\langle\,\rangle \mid P}}{x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty, z : \langle\,\rangle^{\mathbf{H}}/((I^0_0 \mid O^0_0)\ \&\ \mathbf{0}) \vdash_{\mathbf{L}} z(\,).\,\overline{y}\langle\,\rangle \mid P}\ \text{T-Weak}}{x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty, z : \langle\,\rangle^{\mathbf{H}}/((I^0_0 \mid O^0_0)\ \&\ \mathbf{0}) \vdash_{\mathbf{L}} x(\,).\,(z(\,).\,\overline{y}\langle\,\rangle \mid P)}\ \text{T-In}}{x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} (\nu z)\,(x(\,).\,(z(\,).\,\overline{y}\langle\,\rangle \mid P)}\ \text{T-New}$$

On the other hand, the derivation in the restricted type system gets stuck as follows:

$$\dfrac{\dfrac{\cdots}{x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty, z : \langle\,\rangle^{\mathbf{H}}/(I^0_0 \mid O^0_0) \vdash_{\mathbf{L}} z(\,).\,\overline{y}\langle\,\rangle \mid P}}{x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty, z : \langle\,\rangle^{\mathbf{H}}/I^\infty_0 \mid O^\infty_0 \vdash_{\mathbf{L}} x(\,).\,(z(\,).\,\overline{y}\langle\,\rangle \mid P)}\ \text{T-In'}$$

Since $\mathit{rel}(I^\infty_0 \mid O^\infty_0)$ does not hold, we cannot apply T-New. Note, however, that the above example is typed if we move the ν-prefix inside:

$$x : \langle\,\rangle^{\mathbf{L}}/I^\infty_\infty, y : \langle\,\rangle^{\mathbf{L}}/O^\infty_\infty \vdash_{\mathbf{L}} x(\,).\,(\nu z)\,(z(\,).\,\overline{y}\langle\,\rangle \mid P)$$

$$\emptyset \vdash_l^{\mathbf{SD}} \mathbf{0} \quad \text{(ST-Zero)} \qquad \frac{\Gamma_1 \vdash_{l_1}^{\mathbf{SD}} P_1 \qquad \Gamma_2 \vdash_{l_2}^{\mathbf{SD}} P_2 \qquad l \sqsubseteq l_1, l_2}{\Gamma_1 \,|\, \Gamma_2 \vdash_l^{\mathbf{SD}} P_1 \,|\, P_2} \quad \text{(ST-Par)}$$

$$\frac{\Gamma \vdash_{l_2}^{\mathbf{SD}} P \qquad l \sqsubseteq l_1, l_2 \qquad \Gamma(x) = \langle \tilde{\tau} \rangle^{l_1}/U \text{ or } (x \notin dom(\Gamma) \text{ and } U \leq \mathbf{0}) \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1, t_c+1)}(\Gamma \backslash \{x\} \,|\, \tilde{v} : \uparrow\tilde{\tau}) \,|\, x : \langle \tilde{\tau} \rangle^{l_1}/O^0_{t_c}.U \vdash_l^{\mathbf{SD}} \overline{x}\langle \tilde{v} \rangle . P} \quad \text{(ST-Out)}$$

$$\frac{\Gamma \vdash_l^{\mathbf{SD}} P \qquad (\Gamma(x) = \xi/U \wedge rel(U)) \text{ if } x \in dom(\Gamma)}{\Gamma \backslash x \vdash_l^{\mathbf{SD}} (\nu x : \xi)\, P} \quad \text{(ST-New)}$$

$$\frac{\Gamma_i \vdash_{l_i}^{\mathbf{SD}} P_i \text{ (for } i = 1, 2) \qquad \Gamma \leq \Gamma_i \qquad l \sqsubseteq l_i \text{ (for } i = 1, 2)}{\Gamma \,|\, v : \mathbf{bool}^l \vdash_l^{\mathbf{SD}} \mathbf{if}\ v\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2} \quad \text{(ST-If)}$$

$$\frac{\Gamma \vdash_l^{\mathbf{SD}} P \qquad \tau_i \leq \Gamma(y_i) \text{ or } (y_i \notin dom(\Gamma) \text{ and } ob(\tau_i) = \infty) \text{ (for each } i) \qquad t_c = \infty \Rightarrow l_1 \sqsubseteq l_2}{\uparrow^{(t_c+1, t_c+1)}\Gamma \backslash \{x\}, x : \langle \tilde{\tau} \rangle^{l_1}/I^0_{t_c}.U \vdash_l^{\mathbf{SD}} x(\tilde{y}).\, P} \quad \text{(ST-In)}$$

$$\frac{\Gamma \vdash_l^{\mathbf{SD}} P}{*\Gamma \vdash_l^{\mathbf{SD}} *P} \quad \text{(ST-Rep)}$$

Figure 3: Syntax-Directed Typing Rules

The overall structure of the type inference algorithm is similar to other constraint-based type inference algorithms for the π-calculus [13, 14, 18]: Given an input (Γ, P, l), we can first obtain a set $\mathcal{C}$ of constraints on type variables, usage variables, etc. such that $\theta\mathcal{C}$ holds if and only if $\theta\Gamma \vdash_{\theta l} \theta P$ holds. Then, we can reduce $\mathcal{C}$ step by step to decide whether $\mathcal{C}$ is satisfiable.

5.1 Step 1: Extracting constraints based on syntax-directed rules

We can convert the restricted typing rules to syntax-directed ones, by combining each typing rule with T-Weak. For example, the rule for parallel composition becomes:

$$\frac{\Gamma_1 \vdash_{l_1}^{\mathbf{SD}} P_1 \qquad \Gamma_2 \vdash_{l_2}^{\mathbf{SD}} P_2 \qquad l \sqsubseteq l_1, l_2}{\Gamma_1 \,|\, \Gamma_2 \vdash_l^{\mathbf{SD}} P_1 \,|\, P_2}$$

The whole set of syntax-directed rules is given in Figure 3. It is equivalent to the restricted typing rules in the following sense.

Lemma 5.1: If $\Gamma \vdash_l P$, then there exists Γ', l' such that $\Gamma' \vdash_{l'}^{\mathbf{SD}} P$ with $\Gamma \leq \Gamma'$ and $l \sqsubseteq l'$. If $\Gamma \vdash_l^{\mathbf{SD}} P$, then $\Gamma \vdash_l P$.

The required set of constraints can be obtained by using the syntax-directed rules in a standard manner [13, 14, 18]. See Appendix A for more details.

By reducing constraints on types first, we obtain constraints on usages and secrecy/capability/obligation levels of the following forms:

$$\rho \leq U \qquad l_1 \sqsubseteq l_2 \qquad rel(U)$$
$$\eta = \infty \Rightarrow l_1 \sqsubseteq l_2 \qquad l = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(U)$$

Here, $\mathbf{Cap}_\infty(U)$ in the last constraint means that all the capability levels appearing in U are ∞. The constraint comes from the well-formedness condition on types (recall Definition 3.15). The meta-variable l represents $\mathbf{L}$, $\mathbf{H}$, or a variable (called *secrecy variables*) representing an unknown secrecy level. The meta-variable η represents a variable (called *level variables*) representing an unknown obligation/capability level. Usage U in constraints may contain the operation $\uparrow$ (in addition to usage constructors), and all the capability level annotations in U are level variables.

5.2 Reducing constraints

Constraints are reduced step by step as described below.

5.2.1 Solving subusage constraints

A set of subusage constraints $\{\rho_1 \leq U_1, \ldots, \rho_n \leq U_n\}$ is solved in the following two steps:

1. Transform the constraints so that each usage variable appears exactly once in the lefthand side of the inequalities. This is achieved by replacing $\{\rho \leq U_1, \rho \leq U_2\}$ with $\rho \leq U_1 \,\&\, U_2$, and add $\rho \leq \rho$ for each usage variable ρ that does not appear in the lefthand side.

2. Eliminate subusage constraints by repeatedly applying the rule: $\mathcal{C} \cup \{\rho \leq U\} \longrightarrow [\rho \mapsto \mu\rho.U]\mathcal{C}$.

The set of reduced constraints is satisfiable if and only if the original set of constraints is satisfiable, since by Lemma 3.12, $\rho = \mu\rho.U$ is the greatest (with respect to $\leq$) solution for $\rho \leq U$. Note that the constraints other than subusage constraints are monotonic with respect to $\leq$ in the sense that if $U_1 \leq U_2$, then $[\alpha \mapsto U_1]\mathcal{C}$ implies $[\alpha \mapsto U_2]\mathcal{C}$.

5.2.2 Removing the operator $\uparrow$

Usages in the set of constraints obtained so far may contain the operator $\uparrow$. $\uparrow U$ can be replaced by $\uparrow^{(ob_I(U), ob_O(U))} U$. We can show by induction on the structure of U that $ob_\alpha(U)$ is expressed in terms of **min**, **max**, constants (in $\mathbf{Nat} \cup \{\infty\}$), and expressions of the form $\eta + n$ where η is a level variable. The nontrivial is the case where U is of the form $\mu\rho.U_1$. By the definition of ob_α, $ob_\alpha(U) = \mathbf{lfp}(\lambda x.ob_\alpha^{\{\rho \mapsto x\}}(U_1))$. By induction hypothesis, $ob_\alpha^{\{\rho \mapsto x\}}(U_1)$ can be normalized to:

$$\mathbf{max}(e_1, \ldots, e_n)$$

where each e_i is of the form c_i or $\mathbf{min}(x + n_i, c_i)$. Here, $n_i \in \mathbf{Nat}$ and c_i is an expression not containing x. If $n > 0$, then

$$\mathbf{lfp}(\lambda x.\mathbf{max}(\mathbf{min}(x + n, c), e)) = \mathbf{lfp}(\lambda x.\mathbf{max}(c, e))$$

(Note that $\mathbf{max}(\mathbf{min}(x+n, c), e)$ and $\mathbf{max}(c, e)$ can differ only in the range $0 \leq x < c - n$, and that there is no fixpoint in that range.) So, we can assume without loss of generality that $ob_\alpha^{\{\rho \mapsto x\}}(U_1)$ is of the form $\mathbf{max}(c, \mathbf{min}(x, c'))$ or c. In either case, $ob_\alpha(U) = c$.

Example 5.2: $ob_\alpha(\mu\rho.\uparrow\rho) = \mathbf{lfp}(\lambda x.x + 1) = \mathbf{lfp}(\lambda x.\mathbf{max}(\mathbf{min}(x + 1, \infty))) = \mathbf{lfp}(\lambda x.\mathbf{max}(\infty)) = \infty$.

5.2.3 Reducing $rel(U)$

Next, we eliminate constraints of the form $rel(U)$. This step is a little involved: Since it is the same as the corresponding step in the type inference algorithm for our previous type system [18], please refer to [18] for a more comprehensive description of this step. We can assume that the operation $\uparrow$ on usages no longer appears in U.

By definition, $rel(U)$ holds if and only if $ob_\alpha(U') \leq cap_{\overline{\alpha}}(U')$ holds for every U' such that $U \longrightarrow^* U'$. The set of such U' can be infinite, but we can normalize U' by using the lemma below. We write $\cong$ for the least equivalence relation on usages that satisfies the monoid laws on $\mid$ (where $\mathbf{0}$ is the unit) and the laws $\uparrow^{(t_1,t_2)}(\uparrow^{(t_1',t_2')} U) \cong \uparrow^{(\mathbf{max}(t_1,t_1'),\mathbf{max}(t_2,t_2'))} U$ and $\uparrow^{(t_1,t_2)}(U_1 \mid U_2) \cong \uparrow^{(t_1,t_2)} U_1 \mid \uparrow^{(t_1,t_2)} U_2$. We write nU for parallel composition of n occurrences of U. $0U$ is $\mathbf{0}$.

Lemma 5.3: For any U, there exists a finite set of usages $\{U_1, \ldots, U_n\}$ such that for any U' such that $U \longrightarrow^* U'$, there exist $k_1, \ldots, k_n \in \mathbf{Nat}$ such that $U' \cong k_1 U_1 \mid \cdots \mid k_n U_n$.

The finite set of usages in the above lemma is a set of subexpressions of the usages obtained by expanding recursive usages in U.

The following lemma ensures that whether $ob_\alpha(U') \leq cap_{\overline{\alpha}}(U')$ holds depends only on whether the indices $k_1, \ldots, k_n$ are 0 or not, so that we need to check only a finite number of cases.

Lemma 5.4: If $U' \cong k_1U_1 \mid \cdots \mid k_nU_n$ with $k_1, \ldots, k_n > 0$, then $ob_\alpha(U') \leq cap_{\overline{\alpha}}(U')$ if and only if $ob_\alpha(U_1 \mid \cdots \mid U_n) \leq cap_{\overline{\alpha}}(U_1 \mid \cdots \mid U_n)$.

By the decidability of the reachability problem of Petri nets [6], we have the following lemma. (In practice, some approximation is needed when the reachability of the Petri net cannot be decided in a reasonable amount of time.)

Lemma 5.5: Let $\{U_1, \ldots, U_m\}$ be a subset of the set of usages in Lemma 5.3. Then, it is decidable whether there exist $k_1, \ldots, k_m > 0$ such that $U \longrightarrow^* \cong k_1U_1 \mid \cdots \mid k_mU_m$.

Thus, the constraint $rel(U)$ is replaced by the conjunction of constraints of the form $ob_\alpha(U_1 \mid \cdots \mid U_m) \leq cap_{\overline{\alpha}}(U_1 \mid \cdots \mid U_m)$. By the result of the previous subsection, such a constraint can be further reduced to a constraint of the form $t_1 \leq t_2$. Moreover, by the definition of $cap_\alpha(U)$, t_2 can be expressed in the form $\mathbf{min}(\eta_1, \ldots, \eta_n)$ where $\eta_1, \ldots, \eta_n$ are level variables. So, $t_1 \leq t_2$ can be further reduced to $t_1 \leq \eta_1, \ldots, t_1 \leq \eta_n$.

5.2.4 Final step

Now the resulting constraint is a set of primitive constraints of the form:

$$t \leq \eta \quad l_1 \sqsubseteq l_2 \quad \eta = \infty \Rightarrow l_1 \sqsubseteq l_2 \quad l = \mathbf{L} \Rightarrow \eta = \infty$$

Constraints of the first and fourth forms can be solved by a symbolic method. The constraint $l = \mathbf{L} \Rightarrow \eta = \infty$ can be first converted into a constraint $\mathbf{IF}(l = \mathbf{L}, \infty, 0) \leq \eta$ of the first form, where $\mathbf{IF}(e_1, e_2, e_3)$ is e_2 if e_1 holds and it is e_3 otherwise. In each constraint of the first form, t can be normalized to $\mathbf{max}(e_1, \ldots, e_n)$, where each e_i is of the form $\mathbf{min}(\eta + n, c)$ or c, and c does not contain the variable η. Notice that $\mathbf{max}(\mathbf{min}(\eta, c), e) \leq \eta$ if and only if $e \leq \eta$, and that for $n > 0$, $\mathbf{max}(\mathbf{min}(\eta + n, c), e) \leq \eta$ if and only if $\mathbf{max}(c, e) \leq \eta$. So, we can remove η from t in $t \leq \eta$. Thus, we can eliminate the level variable η by substituting t for the occurrences of η in the other constraints.

After all the level variables have been eliminated, the remaining constraints can be normalized to the following set of constraints:

$$\{\delta_1 \sqsubseteq F_1(\delta_1, \ldots, \delta_n),\ \cdots\ \delta_n \sqsubseteq F_n(\delta_1, \ldots, \delta_n),\\ l_1 \sqsubseteq G_1(\delta_1, \ldots, \delta_n),\ \cdots,\ l_m \sqsubseteq G_m(\delta_1, \ldots, \delta_n)\}$$

Here, δ_i is a secrecy variable, and l_j is a constant $\mathbf{L}$ or $\mathbf{H}$. F_i and G_j are monotonic functions on $\delta_1, \ldots, \delta_n$, constructed from $\mathbf{H}$, $\mathbf{L}$, secrecy variables, and $\mathbf{IF}(t = \infty, l, \mathbf{H})$ (where l is a secrecy variable or a constant). Note that each constraint of the form $t = \infty \Rightarrow l_1 \sqsubseteq l_2$ has been normalized to $l_1 \sqsubseteq \mathbf{IF}(t = \infty, l_2, \mathbf{H})$. Here, t is anti-monotonic with respect to secrecy variables, hence $\mathbf{IF}(t = \infty, l_2, \mathbf{H})$ is monotonic on secrecy variables.

Let us abbreviate the above constraints to $\{\vec{\delta} \sqsubseteq \vec{F}(\vec{\delta}), \vec{l} \sqsubseteq \vec{G}(\vec{\delta})\}$. Since $\vec{F}$ is monotonic, we have:

$$\vec{\mathbf{H}} \sqsupseteq \vec{F}(\vec{\mathbf{H}}) \sqsupseteq \vec{F}^2(\vec{\mathbf{H}}) \sqsupseteq \cdots$$

So, we can find $k \in \mathbf{Nat}$ such that $\vec{\delta} = \vec{F}^k(\vec{\mathbf{H}})$ is the greatest solution for $\vec{\delta} \sqsubseteq \vec{F}(\vec{\delta})$. For such k, the constraint $\{\vec{\delta} \sqsubseteq \vec{F}(\vec{\delta}), \vec{l} \sqsubseteq \vec{G}(\vec{\delta})\}$ is satisfiable if and only if $\vec{l} \sqsubseteq \vec{G}(\vec{F}^k(\vec{\mathbf{H}}))$ holds.

5.3 Examples

Example 5.6: Let us consider the following process P:

$$(\nu s : \langle \beta_1 \rangle^{\mathbf{H}}/\rho_1)\,(\nu z : \langle\,\rangle^{l})\,(*s(y)\,.\,(\overline{y}\langle\,\rangle \mid \overline{s}\langle y\rangle) \mid \overline{s}\langle z\rangle \mid z(\,).\,\overline{x}\langle\,\rangle)$$

Let Γ be $x : \langle\,\rangle^{\mathbf{L}}/\rho_2$. Given $(\Gamma, \mathbf{L}, P)$ as an input, we obtain the following constraints. (We have already reduced constraints on types and omitted unimportant constraints.)

$$\begin{array}{lll}
\beta_1 = \langle\,\rangle^{l}/\rho_3 & \mathbf{H} \sqsubseteq l & \eta_5 = \infty \Rightarrow l \sqsubseteq \mathbf{L} \\
\rho_1 \leq (*I^0_{\eta_1}.O^0_{\eta_2}) \mid O^0_{\eta_3} & \rho_2 \leq \uparrow^{(\eta_5+1,\eta_5+1)} O^0_{\eta_6} (= O^{\eta_5+1}_{\eta_6}) & \\
\rho_3 \leq O^0_{\eta_4} \mid \uparrow^{(\eta_2+1,\eta_2+1)} \uparrow \rho_3 & \rho_4 \leq (\uparrow^{(\eta_3+1,\eta_3+1)} \uparrow \rho_3) \mid I^0_{\eta_5} & \\
\mathbf{Cap}_\infty(\rho_2) & \mathit{rel}(\rho_1) & \mathit{rel}(\rho_4)
\end{array}$$

Here, $\rho_1, \rho_2, \rho_3, \rho_4$ are usages of s, x, y, and z respectively. The constraint $\eta_5 = \infty \Rightarrow l \sqsubseteq \mathbf{L}$ comes from typing constraints on $z(\,).\,\overline{x}\langle\,\rangle$. Constraints $\mathbf{H} \sqsubseteq l$ and $\mathbf{Cap}_\infty(\rho_2)$ come from the conditions on well-formed types. By solving the subusage constraints, we obtain the following solutions for $\rho_1, \ldots, \rho_4$:

$$\begin{array}{ll}
\rho_1 = (*I^0_{\eta_1}.O^0_{\eta_2}) \mid O^0_{\eta_3} & \rho_2 = O^{\eta_5+1}_{\eta_6} \\
\rho_3 = \mu\rho.(O^0_{\eta_4} \mid \uparrow^{(\eta_2+1,\eta_2+1)} \uparrow \rho) & \rho_4 = (\uparrow^{(\eta_3+1,\eta_3+1)} \uparrow \rho_3) \mid I^0_{\eta_5}
\end{array}$$

To remove $\uparrow$, we compute $ob_I(\rho_3)$ and $ob_O(\rho_3)$.

$$\begin{array}{rcl}
ob_I(\rho_3) & = & \mathbf{lfp}(\lambda x.\mathbf{min}(\infty, \mathbf{max}(\eta_2 + 1, x + 1))) \\
 & = & \mathbf{lfp}(\lambda x.\mathbf{max}(\eta_2 + 1, \mathbf{min}(x + 1, \infty)) \\
 & = & \mathbf{lfp}(\lambda x.\mathbf{max}(\eta_2 + 1, \infty)) = \infty \\
ob_O(\rho_3) & = & \mathbf{lfp}(\lambda x.\mathbf{min}(0, \mathbf{max}(\eta_2 + 1, x + 1))) \\
 & = & \mathbf{lfp}(\lambda x.0) = 0
\end{array}$$

So, ρ_3 and ρ_4 are: $\mu\rho.(O^0_{\eta_4} \mid \uparrow^{(\infty,\eta_2+1)} \rho)$ and $(\uparrow^{(\infty,\eta_3+1)} \rho_3) \mid I^0_{\eta_5}$. Since $\rho_1 \longrightarrow \rho_1 \longrightarrow \cdots$, $cap_I(\rho_1) = \eta_1$ and $cap_O(\rho_2) = \mathbf{min}(\eta_2, \eta_3)$, $\mathit{rel}(\rho_1)$ is reduced to the constraints $0 \leq \mathbf{min}(\eta_2, \eta_3)$ and $0 \leq \eta_1$ (which are tautologies). On the other hand, ρ_4 can be reduced to $U_1 = \uparrow^{(\infty, \mathbf{max}(\eta_3+1, \eta_2+1))} \rho_3$ or $U_2 = O^{\eta_3+1}_{\eta_4} \mid U_1$. So, $\mathit{rel}(\rho_4)$ can be reduced to

$$\begin{array}{ll}
ob_I(\rho_4) \leq cap_O(\rho_4) & ob_O(\rho_4) \leq cap_I(\rho_4) \\
ob_I(U_i) \leq cap_O(U_i) & ob_O(U_i) \leq cap_I(U_i)\ (i = 1, 2),
\end{array}$$

from which we obtain $\eta_3 + 1 \leq \eta_5$ and $\infty \leq \eta_4$. (We have omitted tautology). So, the remaining constraints are:

$$\begin{array}{lll}
\infty \leq \eta_6 & \eta_3 + 1 \leq \eta_5 & \infty \leq \eta_4 \\
\mathbf{H} \sqsubseteq l & l \sqsubseteq \mathbf{IF}(\eta_5 = \infty, \mathbf{L}, \mathbf{H}) &
\end{array}$$

The least solution for constraints on level variables is: $\eta_1 = \eta_2 = \eta_3 = 0, \eta_5 = \eta_3 + 1 = 1, \eta_4 = \eta_6 = \infty$. So, we obtain the constraints $\{\mathbf{H} \sqsubseteq l, l \sqsubseteq \mathbf{H}\}$, which hold for $l = \mathbf{H}$.

Example 5.7: Let P be the process:

$$(\nu x : \langle\,\rangle^{\mathbf{H}})\,(\nu \mathit{secret} : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}})\,(A \mid C \mid \overline{x}\langle\,\rangle \mid \overline{\mathit{secret}}\langle b\rangle)$$

in Example 3.26 and Γ be $\mathit{public} : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/\rho_1, x : \langle\,\rangle^{\mathbf{L}}/\rho_2$. Given $(\Gamma, \mathbf{L}, P)$ as an input, we obtain the

following constraints.

$$\begin{array}{l}
\rho_1 \le \uparrow^{(\eta_5+1,\eta_5+1)}\uparrow^{(\eta_{15}+1,\eta_{15}+1)} I^0_{\eta_1}.O^0_{\eta_2} \\
\rho_2 \le I^0_{\eta_3}.\uparrow^{(\eta_{10}+1,\eta_{10}+1)} O^0_{\eta_4} \\
\rho_3 \le I^0_{\eta_5}.O^0_{\eta_6} \mid O^0_{\eta_7} \\
\rho_4 \le \uparrow^{(\eta_5+1,\eta_5+1)}(I^0_{\eta_8}.O^0_{\eta_9} \,\&\, \mathbf{0}) \mid \uparrow^{(\eta_3+1,\eta_3+1)} I^0_{\eta_{10}}.O^0_{\eta_{11}} \mid O^0_{\eta_{12}} \\
\rho_5 \le \uparrow^{(\eta_5+1,\eta_5+1)}((\uparrow^{(\eta_8+1,\eta_8+1)} O^0_{\eta_{13}} \,\&\, O^0_{\eta_{14}}) \mid I^0_{\eta_{15}}) \\
rel(\rho_3) \qquad rel(\rho_4) \qquad rel(\rho_5) \\
\mathbf{Cap}_\infty(\rho_1) \qquad \mathbf{Cap}_\infty(\rho_2) \qquad \delta = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(\rho_5) \\
\eta_5 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\
\eta_8 = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta \\
\eta_{15} = \infty \Rightarrow \delta \sqsubseteq \mathbf{L} \\
\eta_{10} = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\
\mathbf{H} \sqsubseteq \delta
\end{array}$$

Here, ρ_3, ρ_4, ρ_5 are usages of channels *secret*, x, and c respectively, and δ is the secrecy level of c.

By solving the subusage constraints, we obtain:

$$\begin{array}{l}
\rho_1 = I^{\mathbf{max}(\eta_5+1,\eta_{15}+1)}_{\eta_1}.O^0_{\eta_2} \\
\rho_2 = I^0_{\eta_3}.O^{\eta_{10}+1}_{\eta_4} \\
\rho_3 = I^0_{\eta_5}.O^0_{\eta_6} \mid O^0_{\eta_7} \\
\rho_4 = (I^{\eta_5+1}_{\eta_8}.O^0_{\eta_9} \,\&\, \mathbf{0}) \mid I^{\eta_3+1}_{\eta_{10}}.O^0_{\eta_{11}} \mid O^0_{\eta_{12}} \\
\rho_5 = (O^{\mathbf{max}(\eta_5+1,\eta_8+1)}_{\eta_{13}} \,\&\, O^{\eta_5+1}_{\eta_{14}}) \mid I^{\eta_5+1}_{\eta_{15}}
\end{array}$$

Therefore, the constraints:

$$\begin{array}{l}
rel(\rho_3) \qquad rel(\rho_4) \qquad rel(\rho_5) \\
\mathbf{Cap}_\infty(\rho_1) \qquad \mathbf{Cap}_\infty(\rho_2) \qquad \delta = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(\rho_5)
\end{array}$$

are reduced to:

$$\begin{array}{l}
\infty \le \eta_6 \\
\eta_3 + 1 \le \eta_{12} \qquad \infty \le \eta_{11} \qquad \infty \le \eta_9 \\
\eta_5 + 1 \le \eta_{13} \qquad \eta_5 + 1 \le \eta_{14} \qquad \mathbf{max}(\eta_5 + 1, \eta_8 + 1) \le \eta_{15} \\
\infty \le \eta_1, \eta_2, \eta_3, \eta_4 \\
\mathbf{IF}(\delta = \mathbf{L}, \infty, 0) \le \eta_{13}, \eta_{14}, \eta_{15}
\end{array}$$

By solving them, we obtain the following solution for level variables:

$$\begin{array}{l}
\eta_1 = \eta_2 = \eta_3 = \eta_4 = \eta_6 = \eta_9 = \eta_{11} = \eta_{12} = \infty \\
\eta_5 = \eta_7 = \eta_8 = \eta_{10} = 0 \\
\eta_{13} = \eta_{14} = \eta_{15} = \mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0))
\end{array}$$

By substituting it for the constraints on secrecy levels, we obtain:

$$\begin{array}{l}
0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\
0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta \\
\mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0)) = \infty \Rightarrow \delta \sqsubseteq \mathbf{L} \\
0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\
\mathbf{H} \sqsubseteq \delta
\end{array}$$

They are normalized the following constraints:

$$\begin{array}{l}
\mathbf{H} \sqsubseteq \mathbf{IF}(0 = \infty, \mathbf{L}, \mathbf{H}) \\
\mathbf{H} \sqsubseteq \mathbf{IF}(0 = \infty, \delta, \mathbf{H}) \\
\delta \sqsubseteq \mathbf{IF}(\mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0)) = \infty, \mathbf{L}, \mathbf{H}) \\
\mathbf{H} \sqsubseteq \mathbf{IF}(0 = \infty, \mathbf{L}, \mathbf{H}) \\
\mathbf{H} \sqsubseteq \delta
\end{array}$$

The constraint is satisfied for $\delta = \mathbf{H}$.

Example 5.8: Let P be the process:

$$(\nu x : \langle\,\rangle^{\mathbf{H}})\,(\nu secret : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}})\,(A \mid B \mid \overline{x}\langle\,\rangle \mid \overline{secret}\langle b \rangle)$$

in Example 3.26 and Γ be $public : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/\rho_1, x : \langle\,\rangle^{\mathbf{L}}/\rho_2$. Given $(\Gamma, \mathbf{L}, P)$ as an input, we obtain the following constraints.

$$\begin{array}{l}
\rho_1 \leq \uparrow^{(\eta_5+1,\eta_5+1)} \uparrow^{(\eta_{15}+1,\eta_{15}+1)} I^0_{\eta_1}.O^0_{\eta_2} \\
\rho_2 \leq \uparrow^{(\eta_3+1,\eta_3+1)} I^0_{\eta_{10}}.O^0_{\eta_{11}} \\
\rho_3 \leq I^0_{\eta_5}.O^0_{\eta_6} \mid O^0_{\eta_7} \\
\rho_4 \leq \uparrow^{(\eta_5+1,\eta_5+1)} (I^0_{\eta_8}.O^0_{\eta_9} \,\&\, \mathbf{0}) \mid I^0_{\eta_3}.\uparrow^{(\eta_{10}+1,\eta_{10}+1)} O^0_{\eta_4} \mid O^0_{\eta_{12}} \\
\rho_5 \leq \uparrow^{(\eta_5+1,\eta_5+1)} ((\uparrow^{(\eta_8+1,\eta_8+1)} O^0_{\eta_{13}} \,\&\, O^0_{\eta_{14}}) \mid I^0_{\eta_{15}}) \\
rel(\rho_3) \qquad rel(\rho_4) \qquad rel(\rho_5) \\
\mathbf{Cap}_\infty(\rho_1) \qquad \mathbf{Cap}_\infty(\rho_2) \qquad \delta = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(\rho_5) \\
\eta_5 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\
\eta_8 = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta \\
\eta_{15} = \infty \Rightarrow \delta \sqsubseteq \mathbf{L} \\
\eta_3 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\
\mathbf{H} \sqsubseteq \delta
\end{array}$$

Here, ρ_3, ρ_4, ρ_5 are usages of channels $secret, x$, and c respectively, and δ is the secrecy level of c. (The difference from the constraints in the previous example is that the usage of ρ_2 and a part of the usage of ρ_4 have been swapped.)

By solving the subusage constraints, we obtain:

$$\begin{array}{l}
\rho_1 = I^{\mathbf{max}(\eta_5+1,\eta_{15}+1)}_{\eta_1}.O^0_{\eta_2} \\
\rho_2 = I^{\eta_3+1}_{\eta_{10}}.O^0_{\eta_{11}} \\
\rho_3 = I^0_{\eta_5}.O^0_{\eta_6} \mid O^0_{\eta_7} \\
\rho_4 = (I^{\eta_5+1}_{\eta_8}.O^0_{\eta_9} \,\&\, \mathbf{0}) \mid I^0_{\eta_3}.O^{\eta_{10}+1}_{\eta_4} \mid O^0_{\eta_{12}} \\
\rho_5 = (O^{\mathbf{max}(\eta_5+1,\eta_8+1)}_{\eta_{13}} \,\&\, O^{\eta_5+1}_{\eta_{14}}) \mid I^{\eta_5+1}_{\eta_{15}}
\end{array}$$

Therefore, the constraints:

$$\begin{array}{l}
rel(\rho_3) \qquad rel(\rho_4) \qquad rel(\rho_5) \\
\mathbf{Cap}_\infty(\rho_1) \qquad \mathbf{Cap}_\infty(\rho_2) \qquad \delta = \mathbf{L} \Rightarrow \mathbf{Cap}_\infty(\rho_5)
\end{array}$$

are reduced to:

$$\begin{array}{l}
\infty \leq \eta_6 \\
\infty \leq \eta_4 \qquad \eta_{10} + 1 \leq \eta_8 \qquad \infty \leq \eta_9 \\
\eta_5 + 1 \leq \eta_{13} \qquad \eta_5 + 1 \leq \eta_{14} \qquad \mathbf{max}(\eta_5 + 1, \eta_8 + 1) \leq \eta_{15} \\
\infty \leq \eta_1, \eta_2, \eta_{10}, \eta_{11} \\
\mathbf{IF}(\delta = \mathbf{L}, \infty, 0) \leq \eta_{13}, \eta_{14}, \eta_{15}
\end{array}$$

By solving them, we obtain the following solution for level variables:

$$\begin{array}{l}
\eta_1 = \eta_2 = \eta_4 = \eta_6 = \eta_8 = \eta_9 = \eta_{10} = \eta_{11} = \eta_{15} = \infty \\
\eta_3 = \eta_5 = \eta_7 = \eta_{12} = 0 \\
\eta_{13} = \eta_{14} = \mathbf{max}(1, \mathbf{IF}(\delta = \mathbf{L}, \infty, 0))
\end{array}$$

By substituting it for the constraints on secrecy levels, we obtain:

$$\begin{array}{l} 0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\ \infty = \infty \Rightarrow \mathbf{H} \sqsubseteq \delta \\ \infty = \infty \Rightarrow \delta \sqsubseteq \mathbf{L} \\ 0 = \infty \Rightarrow \mathbf{H} \sqsubseteq \mathbf{L} \\ \mathbf{H} \sqsubseteq \delta \end{array}$$

From the last constraint, we obtain $\delta = \mathbf{H}$, which contradicts with the third constraint. So, $(\nu x : \langle\,\rangle^{\mathbf{H}})\,(\nu secret : \langle \mathbf{bool}^{\mathbf{H}} \rangle^{\mathbf{H}})\,(A \mid B \mid \overline{x}\langle\,\rangle \mid \overline{secret}\langle b \rangle)$ is not well-typed under any type environment of the form $public : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/\rho_1, x : \langle\,\rangle^{\mathbf{L}}/\rho_2$.

6 Discussions

In Section 3, we have imposed several restrictions on the type system for the sake of simplicity. In this section, we discuss those restrictions and how to remove them. Most of the restrictions (e.g., those on subtyping) are easy to remove although the proof of type soundness and the type inference algorithm becomes a little more complex.

Conditions on well-formed types In Section 3.1, we have presented IN-IDEAL as an ideal rule for input processes. Actually, however, we also need some subtle, extra conditions, which are implicitly enforced by the conditions on well-formed types (Definition 3.15): If x is a high-level channel and P is a low-level process, $x(y).\,P$ is safe only if the success of x is guaranteed by only communications on high-level channels and if y is not used as a low-level value in P. Indeed, without the assumption about well-formed types, the typing rules in Section 3 are unsound. For example, consider the following process (which violates the first condition on well-formed types):

$$\begin{array}{l} (\nu x : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{H}}) \\ (\mathbf{if}\ secret\ \mathbf{then}\ \overline{x}\langle true^{\mathbf{L}} \rangle\ \mathbf{else}\ \overline{x}\langle false^{\mathbf{L}} \rangle \\ \qquad \mid x(y).\,\overline{non\_secret}\langle y \rangle) \end{array}$$

It is well-typed under the type environment:

$$secret : \mathbf{bool}^{\mathbf{H}}, non\_secret : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/O_{\infty}^{\infty}$$

if we remove the first condition on well-formed types, but it leaks the value of *secret*.

Let us also consider the following process:

$$\begin{array}{l} (\nu x : \langle\,\rangle^{\mathbf{H}})\,(\nu y : \langle int^{\mathbf{L}} \rangle^{\mathbf{L}}) \\ \qquad (\mathbf{if}\ secret\ \mathbf{then}\ \overline{x}\langle\,\rangle\ \mathbf{else}\ \mathbf{0} \\ \qquad \mid \overline{y}\langle true^{\mathbf{L}} \rangle \mid x(\,).\,\overline{y}\langle false^{\mathbf{L}} \rangle \\ \qquad \mid y(z).\,(\overline{x}\langle\,\rangle \mid \overline{non\_secret}\langle z \rangle)) \end{array}$$

It is well typed under the type environment:

$$secret : \mathbf{bool}^{\mathbf{H}}, non\_secret : \langle \mathbf{bool}^{\mathbf{L}} \rangle^{\mathbf{L}}/O_{\infty}^{\infty}$$

if we remove the second condition on well-formed types. (Let the usage of y be $O_0^0 \mid O_\infty^\infty \mid I_0^0$ and the usage of x be $(O_\infty^\infty \,\&\, \mathbf{0}) \mid I_1^\infty \mid O_\infty^1$.) The process, however, leaks information about *secret*, since $false^{\mathbf{L}}$ can be sent on *non_secret* only if *secret* is $true^{\mathbf{H}}$.

One way to remove the first condition on well-formed types would be to replace the condition $t_c = \infty \Rightarrow l_1 \sqsubseteq l_2$ in rule T-IN with $(t_c = \infty \vee \mathbf{SL}(\tilde{\tau}) = \mathbf{L}) \Rightarrow l_1 \sqsubseteq l_2$, where $\mathbf{SL}(\tilde{\tau})$ is the least secrecy level

annotation that appears in $\tilde{\tau}$. (We have not yet checked whether this is a sufficient condition.) Then, in the first example above, $\overline{x}\langle \mathit{true}^{\mathbf{L}} \rangle$ is judged to be low-level, so that **if** *secret* **then** $\overline{x}\langle \mathit{true}^{\mathbf{L}} \rangle$ **else** $\overline{x}\langle \mathit{false}^{\mathbf{L}} \rangle$ is ill-typed. The second condition on well-formed types can be removed by changing the conditions $t_c + 1 \leq ob(\Gamma \mid \tilde{v} : \tilde{\tau'})$ and $t_c + 1 \leq ob(\Gamma)$ in T-Out and T-In so that fulfillment of obligations on high-level channels cannot rely on capabilities on low-level channels. (Note that the problem of the second example above was that the output on the high-level channel x on the last line being executed depended on the input capability on the low-level channel y.)

Subtyping Our type system does not allow subtyping based on secrecy levels (e.g., $\mathbf{bool}^{\mathbf{L}} \not\leq \mathbf{bool}^{\mathbf{H}}$). Allowing the relation $\mathbf{bool}^{\mathbf{L}} \leq \mathbf{bool}^{\mathbf{H}}$ does not cause any problem: we only need to redefine the erasure function $\mathbf{Er}_\Gamma(P)$, so that, for example, $\mathit{true}^{\mathbf{L}}$ is also replaced by $\star$ if it is used as a value of type $\mathbf{bool}^{\mathbf{H}}$. On the other hand, introducing subtyping on channel types is tricky. In fact, allowing $\langle \tilde{\tau} \rangle^{\mathbf{L}}/U \leq \langle \tilde{\tau} \rangle^{\mathbf{H}}/U$ for an arbitrary U makes the type system unsound [25]. Honda and Yoshida's type system [12] allows subtyping based on secrecy levels for certain kinds of channels. It is left for future work to find a general condition on U for $\langle \tilde{\tau} \rangle^{\mathbf{L}}/U \leq \langle \tilde{\tau} \rangle^{\mathbf{H}}/U$ to be valid.

Introducing subtyping based on input/output modes [23] does not cause any problem and the type inference algorithm can be developed along the line of [14].

Encoding of functions Unlike Honda and Yoshida's type system [12], there are certain terms of the simply-typed λ-calculus whose termination property cannot be captured in our type system. (For example, our type system cannot guarantee that the process obtained by the call-by-value encoding of **let** $g = \lambda f.f(1)$ **in** $g(\lambda y.g(\lambda x.x))$, which calls g inside g, will eventually return a result.) To remove the limitation, we need to introduce some form of polymorphism on capability/obligation levels. Alternatively, we can treat functions as primitives and give typing rules for them directly.

Treatment of shared variables As discussed in Example 2.9, we can encode operations on shared variables into communication primitives, but the resulting analysis is not precise enough. For example, consider the following command [11]: `if secret>0 then secret := non_secret`. Since reading from the non-secret variable `non_secret` does not leak any information, the command should be safe. However, its encoding into the π-calculus performs communications on non-secret channels, so that it is judged to be unsafe. To overcome this limitation, it seems necessary to take $x(y).(\overline{x}\langle y \rangle \mid P)$ as a primitive, and give a special typing rule for it.[5]

Timing leaks Our type system does not prevent leakage of secret information from the timing behavior. To prevent such leakage, we can use the type system for time-boundedness [15], which can statically guarantee that each communication succeeds in a certain number of reduction steps. An alternative way to avoid timing leaks would be to impose a restriction that programs must be confluent [33].

Ill-typed contexts In Section 4, we considered only well-typed contexts as observers (recall Definition 4.5). In practice, however, we may not be able to assume that malicious processes respect types. To make the non-interference property holds also in the presence of ill-typed processes, we need to extend our type system with a special type for describing untyped values [1, 16, 4].

[5] This is against our goal to uniformly treat various concurrency primitives, but it is inevitable since there is no way to distinguish between $x(n).(\overline{x}\langle n \rangle \mid P)$ and $x(n).(\overline{x}\langle n+1 \rangle \mid P)$ at the type-level: Note that if x is a non-secret channel, the former does not leak any information in the asynchronous π-calculus, while the latter does.

7 Related Work

We have already discussed previous studies on information flow analysis for concurrent programs in Section 1. We discuss some of them in more detail below. Mantel and Sabelfeld [27] have also proposed a type-based information flow analysis for a multi-threaded language with communication primitives. Their analysis suffers from the same problem as Pottier's type system [25] discussed in Section 1. To improve the expressive power, they instead introduced encrypted channels (in addition to high-level/low-level channels) and more communication primitives (such as a primitive for non-blocking receive). Such a solution is orthogonal to our approach, so that we can combine them to obtain a more expressive secure concurrent language. Hennessy and Riely [10, 9] have also studied a type system for the asynchronous π-calculus. Their type system also suffers from the same problem as Pottier's one.

Zdancewic and Myers [33] have recently proposed a type system for a concurrent language having (a restricted form) of join-patterns [7] as synchronization primitives. To overcome the problem discussed in Section 1, their type system introduce linear (use-once) channels and control the temporal ordering on communications on linear channels (which are controlled in our type system through capability/obligation levels) in a syntactic manner. It seems fairly easy to encode their typed calculus into our typed calculus (although extensions with subtyping discussed in Section 6 is necessary). In fact, a join-pattern $\mathbf{let}\ c_1(\tilde{x_1})|\cdots|c_n(\tilde{x_n}) \triangleright P\ \mathbf{in}\ Q$ can be encoded into

$$(\nu c_1)\cdots(\nu c_n)\,(*c_1(\tilde{x_1}).\cdots c_n(\tilde{x_n}).\,P \mid Q)$$

and a linear join-pattern $\mathbf{let}\ c_1(\tilde{x_1})|\cdots|c_n(\tilde{x_n}) \multimap P\ \mathbf{in}\ Q$ can be encoded into

$$(\nu c_1)\cdots(\nu c_n)\,(c_1(\tilde{x_1}).\cdots c_n(\tilde{x_n}).\,P \mid Q)$$

On the other hand, it is not clear how to extend Zdancewic and Myers [33]'s type system to encode our calculus into their calculus. For example, since the success of communications is guaranteed only on linear channels, their type system cannot deal with the example given in Section 1. Moreover, their type system impose restrictions that linear channels cannot be passed through linear channels, etc. (The type system of Honda et al. [11] also imposes similar restrictions.)

Our type system has been obtained by simplifying and refining our previous type system for lock-freedom [15]. The most important technical contribution with respect to the previous work (besides the extension to deal with secrecy[6]) is the development of a sound and complete type inference algorithm (and refinement of the type system to enable the type inference). The algorithm has been inspired from our type inference algorithm for a deadlock-free π-calculus [18]. The latter algorithm was, however, incomplete (see [18] for details). The completeness of the type inference algorithm in this paper has been obtained by careful definitions of the semantics of recursive usages: The key property of recursive usages is that $\mu\rho.U$ is the greatest usage that satisfies $\rho \leq U$.

Our technique for proving non-interference using the erasure function has been inspired from our recent work on type-based useless-code elimination for the π-calculus [16] and is probably also related with Pottier's proof technique [25]. The proof in the present paper is, however, more sophisticated since we need a lock-freedom property to show the correspondence between P and $\mathbf{Er}_\Gamma(P)$.

8 Conclusion

We have presented a general type system for information flow analysis for the pi-calculus and proved its soundness. The generality of the type system enabled precise analysis of information flow and also

[6]The reader might feel that the extension to deal with secrecy are easy after reading Section 3.1, but actually it is also subtle enough as discussed in the paragraph about well-formed types in Section 6.

development of a sound type inference algorithm The algorithm is complete with respect to a slightly less expressive variant of the type system.

Our type system in this paper can also be used for program slicing and useless-code elimination for concurrent programs, since both information flow analysis and program slicing are instances of dependency analysis [2]. Our previous type system for slicing and useless-code elimination for the π-calculus [16] did not take the lock-freedom property into account, so that it was not so effective. We can refine it by using the type system in the present paper.

Extending our type-based information flow analysis to deal with encryption/decryption primitives [3] is also interesting future work.

Acknowledgment

We would like to thank Steve Zdancewic, Martín Abadi, and Eijiro Sumii for useful comments and discussions.

References

[1] M. Abadi. Secrecy by typing in security protocols. *JACM*, 46(5):749–786, 1999.

[2] M. Abadi, A. Banerjee, N. Heintze, and J. G. Rieck. A core calculus of dependency. In *Proc. of POPL*, pages 147–169, 1999.

[3] M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999.

[4] L. Cardelli, G. Ghelli, and A. D. Gordon. Secrecy and group creation. In *Proceedings of CONCUR 2000*, volume 1877 of *LNCS*, pages 365–379. Springer-Verlag, 2000.

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

[6] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Journal of Information Processing and Cybernetics*, 30(3):143–160, 1994.

[7] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. of POPL*, pages 372–385, 1996.

[8] N. Heintze and J. Riecke. The slam calculus: programming with secrecy and integrity. In *Proc. of POPL*, pages 365–377, 1998.

[9] M. Hennessy. The security picalculus and non-interference. *Journal of Logic and Algebraic Programming*, 2003. to appear.

[10] M. Hennessy and J. Riely. Information flow vs. resource access in the information asynchronous pi-calculus. In *Proc. of ICALP 2000*, volume 1853 of *LNCS*. Springer-Verlag, July 2000.

[11] K. Honda, V. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In *Proc. of European Symposium on Programming (ESOP) 2000*, volume 1782 of *LNCS*, pages 180–199. Springer-Verlag, 2000.

[12] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Proc. of POPL*, pages 81–92, 2002.

[13] A. Igarashi and N. Kobayashi. A generic type system for the pi-calculus. *Theor. Comput. Sci.* To appear. A preliminary summary appeared in Proceedings of POPL 2001, pp.128-141.

[14] A. Igarashi and N. Kobayashi. Type reconstruction for linear pi-calculus with I/O subtyping. *Info. Comput.*, 161:1–44, 2000.

[15] N. Kobayashi. A type system for lock-free processes. *Info. Comput.*, 177:122–159, 2002.

[16] N. Kobayashi. Useless code elimination and program slicing for the pi-calculus. In *Proceedings of The First Asian Symposium on Programming Languages and Systems (APLAS'03)*, volume 2895 of *LNCS*, pages 55–72, 2003.

[17] N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. *ACM Trans. Prog. Lang. Syst.*, 21(5):914–947, 1999.

[18] N. Kobayashi, S. Saito, and E. Sumii. An implicitly-typed deadlock-free process calculus. Technical Report TR00-01, Dept. Info. Sci., Univ. of Tokyo, January 2000. Available from `http://www.kb.cs.titech.ac.jp/~kobayasi/`. A summary has appeared in Proceedings of CONCUR 2000, Springer LNCS1877, pp.489-503, 2000.

[19] N. Kobayashi and K. Shirane. Type-based information flow analysis for a low-level language. *Computer Software*, 20(2):2–21, 2003. In Japanese. A summary written in English is available from `http://www.kb.cs.titech.ac.jp/~kobayasi/`.

[20] N. Kobayashi and A. Yonezawa. Towards foundations for concurrent object-oriented programming – types and language design. *Theory and Practice of Object Systems*, 1(4):243–268, 1995.

[21] R. Milner. The polyadic π-calculus: a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer-Verlag, 1993.

[22] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[23] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[24] B. C. Pierce and D. N. Turner. Concurrent objects in a process calculus. In *Theory and Practice of Parallel Programming (TPPP), Sendai, Japan (Nov. 1994)*, volume 907 of *LNCS*, pages 187–215. Springer-Verlag, 1995.

[25] F. Pottier. A simple view of type-secure information flow in the π-calculus. In *Proceedings of the 15th IEEE Computer Security Foundations Workshop*, pages 320–330, 2002.

[26] F. Pottier and V. Simonet. Information flow inference for ML. In *Proc. of POPL*, pages 319–330, 2002.

[27] A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In *Proceedings of the 9th International Static Analysis Symposium*, LNCS 2477, pages 376–394, Madrid, Spain, September 2002. Springer-Verlag.

[28] D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[29] G. Smith and D. Volpano. Secure information flow in a multi-threaded imperative language. In *Proc. of POPL*, pages 355–364, 1998.

[30] E. Sumii and N. Kobayashi. A generalized deadlock-free process calculus. In *Proc. of Workshop on High-Level Concurrent Language (HLCL'98)*, volume 16(3) of *ENTCS*, pages 55–77, 1998.

[31] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[32] S. Zdancewic and A. C. Myers. Secure information flow via linear continuations. *Higher-Order and Symbolic Computation*, 15(2/3):209–234, 2002.

[33] S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *Proceedings of the 16th IEEE Computer Security Foundations Workshop*, 2003.

A Algorithm for extracting constraints

An algorithm *Tinf* for extracting constraints is shown in Figure 4. In the figure, $\mathcal{C}^{\leq}(\Gamma_1, \Gamma_2)$ denotes the set $\{\Gamma_1(x) \leq \Gamma_2(x) \mid x \in dom(\Gamma_2)\} \cup \{noob(\Gamma_1(x)) \mid x \notin dom(\Gamma_2)\}$ of constraints on types. The constraint $noob(\tau)$ in ST-In denotes the same constraint as $ob(\tau) = \infty$, but $noob(\langle\tilde{\tau}\rangle^l/U)$ should be reduced to $U \leq \mathbf{0}$ rather than $ob(U) = \infty$ (since we want to generate only restricted forms of constraints to simplify the reduction of constraints in the next step). The constraint $WF(\Gamma)$ means that $\Gamma(x)$ is well-formed for every $x \in dom(\Gamma)$. $WF(\tau)$ means that τ is well-formed. Meta-variables β, δ, and η represent variables denoting unknown types, secrecy levels, and capability/obligation levels respectively.

The sub-algorithm PT takes P as an input, and outputs a triple $(\Gamma', l', \mathcal{C}')$ that satisfies: (i) $\theta\Gamma' \vdash_{\theta l'} \theta P$ holds for any substitution θ such that $\theta\mathcal{C}'$ holds, and (ii) if $\Gamma'' \vdash_{l''} \theta'' P$, then there exists a substitution θ such that $\theta\mathcal{C}'$, $\Gamma'' \leq \theta\Gamma'$, $l'' \sqsubseteq \theta l'$, and $\theta'' P = \theta P$.

Let $Tinf(\Gamma, l, P) = \mathcal{C}$. Then, $\theta\mathcal{C}$ holds if and only if $\theta\Gamma \vdash_{\theta l} \theta P$ holds.

B Miscellaneous definitions

This section introduces miscellaneous definitions used for proving theorems.

We first define the relation $\sim$ on types, Intuitively, $\tau \sim \tau'$ holds when τ and τ' are identical except for their outermost usages.

Definition B.1: The relation $\tau \sim \tau'$ is the least equivalence relation satisfying the rules: $\mathbf{unit} \sim \mathbf{unit}$, $\mathbf{bool}^l \sim \mathbf{bool}^l$ and $\langle\tau_1, \ldots, \tau_n\rangle^l/U \sim \langle\tau_1, \ldots, \tau_n\rangle^l/U'$ for $l \in \{\mathbf{H}, \mathbf{L}\}$. The relation $\sim$ is extended to a relation on type environments by: $\Gamma \sim \Gamma'$ if and only if $dom(\Gamma) = dom(\Gamma')$ and $\Gamma(x) \sim \Gamma'(x)$ for every $x \in dom(\Gamma)$.

Definition B.2: $ob(\tau)$ is defined by:

$$ob(\mathbf{unit}) = ob(\mathbf{bool}^l) = \infty$$
$$ob(\xi/U) = ob(U)$$

Definition B.3 [size of process]: The size of a process P, written $\#(P)$, is defined by:

$$\#(\mathbf{0}) = 0$$
$$\#(\overline{x}\langle\tilde{v}\rangle.\, P) = \#(x(\tilde{y}).\, P) = \#(P) + 1$$
$$\#(*P) = \#((\nu x : \xi)\, P) = \#(P) + 1$$
$$\#(P \mid Q) = \#(\mathbf{if}\ v\ \mathbf{then}\ P\ \mathbf{else}\ Q) = \#(P) + \#(Q) + 1$$

Definition B.4 [strong barbs]: The *strong barbs* of P, written $SBarbs(P)$, is defined by:

$$\begin{aligned} SBarbs(P) \quad = \quad & \{\overline{x} \mid P \preceq (\nu\tilde{y})\,(\overline{x}\langle\tilde{v}\rangle.\, Q \mid R), x \notin \{\tilde{y}\}\} \\ & \cup \{x \mid P \preceq (\nu\tilde{y})\,(x(\tilde{z}).\, Q \mid R), x \notin \{\tilde{y}\}\} \end{aligned}$$

$$
\begin{array}{l}
\mathit{Tinf}(\Gamma, l, P) = \\
\quad \textbf{let } (\Gamma', l', \mathcal{C}') = PT(P) \\
\quad \textbf{in } \mathcal{C}' \cup \mathcal{C}^{\leq}(\Gamma, \Gamma') \cup \{l \sqsubseteq l'\} \cup WF(\Gamma) \\
\\
\\
PT(\mathbf{0}) = (\emptyset, \delta, \emptyset) \\
\qquad\quad \text{(where } \delta \text{ is fresh)} \\
\\
PT(\overline{x}\langle v_1, \ldots, v_n\rangle . P_0) = \\
\quad \textbf{let } (\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0) \\
\qquad \mathcal{C} = \mathcal{C}_0 \cup \{\delta \sqsubseteq \delta_1, \delta \sqsubseteq l_0, \eta = \infty \Rightarrow \delta_1 \sqsubseteq l_0\} \\
\qquad\quad \cup \{\Gamma_0(x) = \langle \beta_1, \ldots, \beta_n\rangle^{\delta_1}/\rho \mid x \in dom(\Gamma_0)\} \cup \{\rho \leq \mathbf{0} \mid x \notin dom(\Gamma_0)\} \\
\quad \textbf{in } (\uparrow^{(\eta+1,\eta+1)}(\Gamma_0 \backslash \{x\} \mid v_1 : \uparrow\beta_1 \mid \cdots \mid v_n : \uparrow\beta_n) \mid x : \langle\tilde{\beta}\rangle^{\delta_1}/O^0_\eta.\rho, \delta, \mathcal{C} \cup \{\, WF(\langle\tilde{\beta}\rangle^{\delta_1}/O^0_\eta.\rho)\}) \\
\quad \text{(where } \beta_1, \ldots, \beta_n, \delta, \delta_1, \eta, \rho \text{ are fresh)} \\
\\
PT(x(y_1, \ldots, y_n) . P_0) = \\
\quad \textbf{let } (\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0) \\
\qquad \mathcal{C} = \mathcal{C}_0 \cup \{\delta \sqsubseteq \delta_1, \delta \sqsubseteq l_0, \eta = \infty \Rightarrow \delta_1 \sqsubseteq l_0\} \\
\qquad\quad \cup \{\Gamma_0(x) = \langle \beta_1, \ldots, \beta_n\rangle^{\delta_1}/\rho \mid x \in dom(\Gamma_0)\} \cup \{\rho \leq \mathbf{0} \mid x \notin dom(\Gamma_0)\} \\
\qquad\quad \cup \{\beta_i \leq \Gamma_0(y_i) \mid y_i \in dom(\Gamma_0)\} \cup \{noob(\beta_i) \mid y_i \notin dom(\Gamma_0)\} \\
\quad \textbf{in } ((\uparrow^{(\eta+1,\eta+1)}\Gamma_0 \backslash \{x\}, x : \langle\tilde{\beta}\rangle^{\delta_1}/I^0_\eta.\rho), \delta, \mathcal{C} \cup \{\, WF(\langle\tilde{\beta}\rangle^{\delta_1}/I^0_\eta.\rho)\}) \\
\quad \text{(where } \beta_1, \ldots, \beta_n, \delta, \delta_1, \eta, \rho \text{ are fresh)} \\
\\
PT(P_1 \mid P_2) = \\
\quad \textbf{let } \ (\Gamma_1, l_1, \mathcal{C}_1) = PT(P_1) \\
\qquad (\Gamma_2, l_2, \mathcal{C}_2) = PT(P_2) \\
\quad \textbf{in } (\Gamma_1 \mid \Gamma_2, \delta, \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{\delta \sqsubseteq l_1, \delta \sqsubseteq l_2\}) \\
\\
PT((\nu x : \xi)\, P_0) = \\
\quad \textbf{let } (\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0) \\
\quad \textbf{in } (\Gamma_0 \backslash \{x\}, l_0, \mathcal{C}_0 \cup \{\Gamma_0(x) = \xi/\rho, rel(\rho) \mid x \in dom(\Gamma_0)\} \cup \{\, WF(\xi/\mathbf{0})\}) \\
\quad \text{(where } \rho \text{ is fresh)} \\
\\
PT(\textbf{if } v \textbf{ then } P_1 \textbf{ else } P_2) = \\
\quad \textbf{let } \ (\Gamma_1, l_1, \mathcal{C}_1) = PT(P_1) \\
\qquad (\Gamma_2, l_2, \mathcal{C}_2) = PT(P_2) \\
\qquad \Gamma = x_1 : \beta_1, \ldots, x_n : \beta_n \text{ where } \{x_1, \ldots, x_n\} = dom(\Gamma_1) \cup dom(\Gamma_2) \text{ and } \beta_1, \ldots, \beta_n \text{ are fresh} \\
\quad \textbf{in } (\Gamma \mid v : \mathbf{bool}^\delta, \delta, \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{\delta \sqsubseteq l_1, \delta \sqsubseteq l_2\} \cup \mathcal{C}^{\leq}(\Gamma, \Gamma_1) \cup \mathcal{C}^{\leq}(\Gamma, \Gamma_2)) \\
\quad \text{(where } \delta \text{ is fresh)} \\
\\
PT(*P_0) = \\
\quad \textbf{let } (\Gamma_0, l_0, \mathcal{C}_0) = PT(P_0) \\
\quad \textbf{in } (*\Gamma_0, l_0, \mathcal{C}_0)
\end{array}
$$

Figure 4: A Type Inference Algorithm

For the sake of technical convenience, we sometimes assume that each input/output process is annotated with its security level and capability level:

$$P \quad ::= \quad \overline{x}\langle v_1, \ldots, v_n\rangle^{l,t}.\, P \mid x(y_1, \ldots, y_n)^{l,t}.\, P \mid \cdots$$

The typing rules for annotated input/output processes are given as follows. (The only change from T-Out and T-In is that the processes are annotated with a security level and a capability level.)

$$\frac{\begin{array}{cc}\Gamma, x : \langle\tilde{\tau}\rangle^{l_1}/U \vdash_{l_2} P & l \sqsubseteq l_1, l_2 \qquad \tilde{\tau'} \leq \uparrow\tilde{\tau} \\ t_c + 1 \leq ob(\Gamma \mid \tilde{v} : \tilde{\tau'}) & t_c = \infty \Rightarrow l_1 \sqsubseteq l_2\end{array}}{\Gamma \mid \tilde{v} : \tilde{\tau'} \mid x : \langle\tilde{\tau}\rangle^{l_1}/O^0_{t_c}.U \vdash_l \overline{x}\langle\tilde{v}\rangle^{l_1,t_c}.\, P} \qquad \text{(T-Out")}$$

$$\frac{\begin{array}{cc}\Gamma, x : \langle\tilde{\tau}\rangle^{l_1}/U, \tilde{y} : \tilde{\tau} \vdash_{l_2} P & l \sqsubseteq l_1, l_2 \\ t_c + 1 \leq ob(\Gamma) & t_c = \infty \Rightarrow l_1 \sqsubseteq l_2\end{array}}{\Gamma, x : \langle\tilde{\tau}\rangle^{l_1}/I^0_{t_c}.U \vdash_l x(\tilde{y})^{l_1,t_c}.\, P} \qquad \text{(T-In")}$$

We annotate the reduction relation with a security level. Intuitively, $P \longrightarrow_l Q$ means that P is reduced to Q by communication on a channel whose security level is l.

Definition B.5: The relation $\longrightarrow_l$ is defined by:

$$\overline{x}\langle\tilde{v}\rangle^{l,t_1}.\, P \mid x(\tilde{y})^{l,t_2}.\, Q \longrightarrow_l P \mid [\tilde{y} \mapsto \tilde{v}]Q \qquad \text{(R-Com)}$$

$$\frac{P \longrightarrow_l Q}{P \mid R \longrightarrow_l Q \mid R} \qquad \text{(R-Par)}$$

$$\frac{P \longrightarrow_l Q}{(\nu x : \xi)\, P \longrightarrow_l (\nu x : \xi)\, Q} \qquad \text{(R-New)}$$

$$\frac{P \preceq P' \qquad P' \longrightarrow_l Q' \qquad Q' \preceq Q}{P \longrightarrow_l Q} \qquad \text{(R-SPCong)}$$

Note that if a process P is well-typed, $P \longrightarrow Q$ holds if and only if $P \longrightarrow_l Q$ holds for some l.

We refine the definition of contexts, so that a hole is annotated with a substitution and a set of variables. The substitution is applied when a process is put into the hole.

Definition B.6 [extended contexts]: The set of *extended contexts* is given by:

$$\begin{array}{rcl} C & ::= & [\,]_{\theta,S} \mid \overline{x}\langle\tilde{v}\rangle^{l,t}.\, C \mid x(\tilde{y})^{l,t}.\, C \\ & & \mid (P \mid C) \mid (C \mid P) \mid *C \mid (\nu x : \xi)\, C \\ & & \mid \textbf{if } v \textbf{ then } C \textbf{ else } P \mid \textbf{if } v \textbf{ then } P \textbf{ else } C \end{array}$$

Here, θ ranges over the set of substitutions, and S ranges over the powerset of variables. If the hole in the context C is $[\,]_{\theta,S}$ and $FV(P) \subseteq S$, $C[P]$ is the process obtained by replacing $[\,]_{\theta,S}$ with θP.

We extend the relations $\preceq$ and $\longrightarrow_l$ on processes to relations on contexts by defining $FV([\,]_{\theta,S})$ to be $\theta S = \{\theta x \mid x \in S\} \cap \mathbf{Var}$, and defining the substitution for the hole by: $\theta_1[\,]_{\theta_2,S} = [\,]_{\theta_1 \circ \theta_2, S}$ (where $\theta_1 \circ \theta_2$ is the composition of substitutions). For example, $\overline{x}\langle v\rangle.\, P \mid x(y).\, [\,]_{id,\{y\}} \longrightarrow P \mid [\,]_{[y \mapsto v]\{y\}}$ where id is the identity substitution. Note that if $C \longrightarrow_l C'$ and $C[P]$ is well defined, then $C[P] \longrightarrow_l C'[P]$ holds. We also define $\mathbf{Er}_\Gamma(C)$ by adding the clause $\mathbf{Er}_\Gamma([\,]_{\theta,S}) = [\,]_{\mathbf{Er}_\Gamma(\theta),S}$ where $\mathbf{Er}_\Gamma([x_1 \mapsto v_1, \ldots, x_n \mapsto v_n]) = [x_1 \mapsto \mathbf{Er}_\Gamma(v_1), \ldots, x_n \mapsto \mathbf{Er}_\Gamma(v_n)]$.

We introduce two subsets of (extended) contexts: the set of finite-level contexts and the set of evaluation contexts. A *finite-level context* is a context whose hole is guarded only by input/output prefixes with finite capability levels. An *evaluation context* is a context whose hole is not guarded by any input/output prefixes.

Definition B.7 [finite-level context]: The set of *finite-level contexts* is given by:

$$\begin{array}{rcl} C & ::= & [\,]_{\theta,S} \mid \overline{x}\langle\tilde{v}\rangle^{\mathbf{H},n}.\,C \mid x(\tilde{y})^{\mathbf{H},n}.\,C \\ & & \mid (P \,|\, C) \mid (C \,|\, P) \mid (\nu x\!:\!\xi)\,C \end{array}$$

Here, n ranges over **Nat**.

Definition B.8 [depth of context, evaluation context]: Let C be a finite level context. The *depth* of C, written $\mathit{depth}(C)$, is defined by:

$$\begin{array}{l} \mathit{depth}([\,]_{\theta,S}) = 0 \\ \mathit{depth}(\overline{x}\langle\tilde{v}\rangle^{n}.\,C) = \mathit{depth}(C) + 1 \\ \mathit{depth}(x(\tilde{y})^{n}.\,C) = \mathit{depth}(C) + 1 \\ \mathit{depth}(P \,|\, C) = \mathit{depth}(C \,|\, P) = \mathit{depth}((\nu x\!:\!\xi)\,C) = \mathit{depth}(C) \end{array}$$

A finite level context whose depth is 0 is called an *evaluation context*.

A *context with two holes* is a term obtained from a process by replacing one sub-process with $[\,]^{(1)}_{\theta_1,S_1}$ and another sub-process with $[\,]^{(2)}_{\theta_2,S_2}$. If C is a context with two holes, we write $C[P_1,P_2]$ for the process obtained from C by replacing $[\,]^{(1)}_{\theta_1,S_1}$ and $[\,]^{(2)}_{\theta_2,S_2}$ with $\theta_1 P_1$ and $\theta_2 P_2$ respectively. *Finite-level contexts with two holes* and *evaluation contexts with two holes* are defined in a similar manner.

Definition B.9: Let C be an extended context. $\mathit{ext}(\Gamma, C)$ is the type environment defined by:

$$\begin{array}{l} \mathit{ext}(\Gamma, [\,]_{\theta_S}) = \Gamma \\ \mathit{ext}(\Gamma, \overline{x}\langle\tilde{v}\rangle^{t}.\,C) = \mathit{ext}(\Gamma, C) \\ \mathit{ext}(\Gamma, x(\tilde{y})^{t}.\,C) = \mathit{ext}((\Gamma, \tilde{y}\!:\!\tilde{\tau}), C) \ (\text{if } \Gamma(x) = \langle\tilde{\tau}\rangle^{l}/U) \\ \mathit{ext}(\Gamma, P \,|\, C) = \mathit{ext}(\Gamma, C \,|\, P) = \mathit{ext}(\Gamma, {*}C) = \mathit{ext}(\Gamma, C) \\ \mathit{ext}(\Gamma, (\nu x\!:\!\xi)\,C) = \mathit{ext}((\Gamma, x\!:\!\xi/\mathbf{0}), C) \\ \mathit{ext}(\Gamma, \textbf{if } v \textbf{ then } C \textbf{ else } P) = \mathit{ext}(\Gamma, C) \\ \mathit{ext}(\Gamma, \textbf{if } v \textbf{ then } P \textbf{ else } C) = \mathit{ext}(\Gamma, C) \end{array}$$

Intuitively, $\mathit{ext}(\Gamma, C)$ is the type environment obtained by adding to Γ bindings on the variables bound by C. Note that if the hole in C is of the form $[\,]_{id,S}$, then $\mathbf{Er}_\Gamma(C[P]) = \mathbf{Er}_\Gamma(C)[\mathbf{Er}_\Delta(P)]$ where $\Delta = \mathit{ext}(\Gamma, C)$. If C is a context with two holes, we write $\mathit{ext}^{(1)}(\Gamma, C)$ and $\mathit{ext}^{(2)}(\Gamma, C)$ for $\mathit{ext}(\Gamma, C[[\,]_{\theta,S}, \mathbf{0}])$ and $\mathit{ext}(\Gamma, C[\mathbf{0}, [\,]_{\theta,S}])$ respectively.

C Proof of Theorem 4.1

Lemma C.1:

1. If $U \preceq U'$, then $U \leq U'$.
2. $U \,|\, \mathbf{0} \leq U$
3. If $\mathit{ob}(U) = \infty$, then $U \leq \mathbf{0}$.
4. If $\mathit{ob}(U_2) < t_o$ and $\mathit{ob}(U_2) < t_c$, then $\alpha^{t_o}_{t_c}.U_1 \,|\, U_2 \leq \alpha^{t_o}_{t_c}.(U_1 \,|\, U_2)$.

5. If $ob(U) = \infty$, then $U' \mid U \leq U'$ for any U'.

6. If $t'_o \leq t_o$ and $t_c \leq t'_c$, then $\alpha^{t_o}_{t_c}.U \leq \alpha^{t'_o}_{t'_c}.U$ holds

7. If $U_1 \leq [\rho \mapsto U_1]U$, then $U_1 \leq \mu\rho.U$.

Proof: The first law can be proved by showing that the relation $\{([\rho \mapsto U]U_1, [\rho \mapsto U']U_1) \mid U \preceq U', FV(U) = \{\rho\}\}$ satisfies the conditions of Definition 3.11. The other laws can be proved in a similar manner. The fifth law follows from the first and second laws by: $U' \mid U \leq U' \mid \mathbf{0} \leq U'$. □

Lemma C.2: If $\Gamma \leq \Gamma'$ and $\Gamma' \longrightarrow \Delta'$, then there exists Δ such that $\Gamma \longrightarrow \Delta$ and $\Delta \leq \Delta'$.

Proof: By the definition of the subusage relation, if $U \leq U'$ and $U' \longrightarrow V'$, then there exists V such that $U \longrightarrow V$ and $V \leq V'$. So, the lemma follows immediately. □

Lemma C.3: If $rel(U)$ and $U \longrightarrow U'$, then $rel(U')$. If $rel(U)$ and $U \leq U'$, then $rel(U')$.

Proof: The first property follows immediately from the definition of *rel*. The second one follows immediately from the definition of $\leq$. □

As a corollary of the above lemma, we obtain the following property.

Lemma C.4: If $closed(\Gamma)$ and $\Gamma \longrightarrow^* \leq \Delta$, then $closed(\Delta)$.

Proof: This follows immediately from Lemma C.3. □

Lemma C.5: If $\Gamma, x : \tau \vdash_l P$ and $x \notin FV(P)$, then $ob(\tau) = \infty$ and $\Gamma \vdash_l P$.

Proof: Straightforward induction on derivation of $\Gamma, x : \tau \vdash_l P$. □

Lemma C.6: If $\Gamma \vdash_l P$ and $P \preceq Q$, then $\Gamma \vdash_l Q$.

Proof: The proof proceeds by induction on derivation of $P \preceq Q$. We show only main cases. The other cases are trivial.

- Case for S-New: There are two cases to consider.
  - Case where $P = (\nu x : \xi)\, P_1 \mid P_2$ and $Q = (\nu x : \xi)\, (P_1 \mid P_2)$: Suppose $\Gamma \vdash_l P$. Then there exist Γ_1, Γ_2, and U such that:
    $$\begin{array}{l} \Gamma_1, x : \xi/U \vdash_l P_1 \\ \Gamma_2 \vdash_l P_2 \\ \Gamma \leq \Gamma_1 \mid \Gamma_2 \\ rel(U) \end{array}$$
    We can assume without loss of generality that $x \notin dom(\Gamma_2)$. So, we have $(\Gamma_1 \mid \Gamma_2), x : \xi/U \vdash_l P_1 \mid P_2$, from which $\Gamma \vdash_l Q$ follows.
  - Case where $P = (\nu x : \xi)\, (P_1 \mid P_2)$ and $Q = (\nu x : \xi)\, P_1 \mid P_2$: Suppose $\Gamma \vdash_l P$. Then there exist Γ_1, Γ_2, and U such that:
    $$\begin{array}{l} \Gamma_1 \vdash_l P_1 \\ \Gamma_2 \vdash_l P_2 \\ \Gamma \leq (\Gamma_1 \mid \Gamma_2) \backslash \{x\} \\ rel((\Gamma_1 \mid \Gamma_2)(x)) \end{array}$$
    If $x \notin dom(\Gamma_2)$, then the result follows immediately. If $\Gamma_2 = \xi/U_2$, then by Lemma C.5, $\Gamma_2 \backslash \{x\} \vdash_l P_2$ and $ob(U_2) = \infty$. So, by Lemma C.1, we can apply T-Sub to $\Gamma_1 \vdash_l P_1$ and obtain $\Gamma_1 \mid x : \xi/U_2 \vdash_l P_1$. By using T-New, T-Par, and T-Sub, we obtain $\Gamma \vdash_l Q$ as required.

- Case for S-IfT: In this case, $P = \mathbf{if}\ \mathit{true}^{l'}\ \mathbf{then}\ Q\ \mathbf{else}\ Q'$. $\Gamma \vdash_l Q$ follows immediately from the typing rules.
- Case for S-Rep: In this case, $P = *P_1$ and $Q = *P_1 \mid P_1$. If $\Gamma \vdash_l P$, then there exists Γ_1 such that $\Gamma_1 \vdash_l P_1$ and $\Gamma \leq *\Gamma_1$. Since $\Gamma \leq *\Gamma_1 \leq *\Gamma_1 \mid \Gamma_1$, we obtain $\Gamma \vdash_l Q$ by using T-Rep, T-Par, and T-Sub.

□

Lemma C.7: If $\Gamma \leq \Delta$ and $[x \mapsto v]\Gamma$ is well defined, then $[x \mapsto v]\Delta$ is also well defined and $[x \mapsto v]\Gamma \leq [x \mapsto v]\Delta$ holds.

Proof: $[x \mapsto v]\Delta$ is well defined since for every $y \in \mathit{dom}(\Delta)$, $\Delta(y) \sim \Gamma(y)$ holds. $[x \mapsto v]\Gamma \leq [x \mapsto v]\Delta$ follows from the fact that $U_1 \leq U_1'$ and $U_2 \leq U_2'$ imply $U_1 \mid U_2 \leq U_1' \mid U_2'$. □

Lemma C.8 [substitution lemma]: If $\Gamma, x : \tau \vdash_l P$ and $[x \mapsto v]\Gamma$ is well defined, then $[x \mapsto v]\Gamma \vdash_l [x \mapsto v]P$ holds.

Proof: This follows by induction on the structure of P. We show only the case where P is an output process. The other cases are similar or trivial.

Suppose $P = \overline{y}\langle \tilde{w} \rangle P_1$. Then, the following conditions must hold:

$$\begin{array}{l} \Gamma_1, y : \langle \tilde{\sigma} \rangle^{l_1}/U \vdash_{l_2} P_1 \\ l \sqsubseteq l_1, l_2 \\ \tilde{\sigma'} \leq \uparrow \tilde{\sigma} \\ t_c < \mathit{ob}(\Gamma_1 \mid \tilde{w} : \tilde{\sigma'}) \\ t_c = \infty \Rightarrow l_1 \sqsubseteq l_2 \\ \Gamma \leq l \sqsubseteq l_2 \Gamma_1 \mid \tilde{w} : \tilde{\sigma'} \mid y : \langle \tilde{\sigma} \rangle^{l_1}/O^0_{t_c}.U \end{array}$$

From the last condition and Lemma C.7, it follows that $[x \mapsto v]\Gamma_1 \mid [x \mapsto v]y : \langle \tilde{\sigma} \rangle^{l_1}/U$ is well defined. So, by the induction hypothesis, we have $[x \mapsto v]\Gamma_1 \mid [x \mapsto v]y : \langle \tilde{\sigma} \rangle^{l_1}/U \vdash_{l_2} [x \mapsto v]P_1$.

We perform case analysis on $[x \mapsto v]y$. If $[x \mapsto v]y = v$ (i.e., $y = x$ or $y = v$), then v must be a variable. We can assume without loss of generality that $\Gamma_1 = \Gamma_1', x : \langle \tilde{\sigma} \rangle^{l_1}/U'$ (since we can add the binding $x : \langle \tilde{\sigma} \rangle^{l_1}/\mathbf{0}$ if $x \notin \mathit{dom}(\Gamma_1)$). Moreover, by the condition $t_c < \mathit{ob}(\Gamma_1 \mid \tilde{w} : \tilde{\sigma'})$, it must be the case that $t_c < \mathit{ob}(U')$. By using T-Out, we obtain:

$$[x \mapsto v]\Gamma_1' \mid \tilde{w'} : \tilde{\sigma'} \mid v : \langle \tilde{\sigma} \rangle^{l_1}/O^0_{t_c}.(U' \mid U) \vdash_{l_2} [x \mapsto v]P$$

where $\tilde{w'} = [x \mapsto v]\tilde{w}$. We obtain $[x \mapsto v]\Gamma \vdash_l [x \mapsto v]P$ by using T-Sub, since

$$\begin{array}{rcl} [x \mapsto v]\Gamma & \leq & [x \mapsto v](\Gamma_1 \mid \tilde{w} : \tilde{\sigma'} \mid y : \langle \tilde{\sigma} \rangle^{l_1}/O^0_{t_c}.U) \\ & \leq & [x \mapsto v]\Gamma_1' \mid \tilde{w'} : \tilde{\sigma'} \mid v : \langle \tilde{\sigma} \rangle^{l_1}/(U' \mid O^0_{t_c}.U) \\ & \leq & [x \mapsto v]\Gamma_1' \mid \tilde{w'} : \tilde{\sigma'} \mid v : \langle \tilde{\sigma} \rangle^{l_1}/O^0_{t_c}.(U' \mid U) \end{array}$$

The last relation uses the fourth law of Lemma C.1.

If $[x \mapsto v]y \neq v$ (i.e., $y \neq x$ and $y \neq v$, which also imply $[x \mapsto v]y = y$), we have $[x \mapsto v]\Gamma_1, y : \langle \tilde{\sigma} \rangle^{l_1}/U \vdash_{l_2} [x \mapsto v]P_1$. By applying T-Out, we obtain $[x \mapsto v]\Gamma_1 \mid \tilde{w'} : \tilde{\sigma'} \mid y : \langle \tilde{\sigma} \rangle^{l_1}/O^0_{t_c}.U \vdash_{l_2} [x \mapsto v]P$ where $\tilde{w'} = [x \mapsto v]\tilde{w}$. By using T-Sub, we get $[x \mapsto v]\Gamma \vdash_l [x \mapsto v]P$ as required. □

Proof of Theorem 4.1: The proof proceeds by induction on derivation of $P \longrightarrow Q$, with case analysis on the last rule used.

- Case for R-Com: In this case, $P = \overline{x}\langle\tilde{v}\rangle.\, P_1 \mid x(\tilde{y}).\, P_2$ and $Q = P_1 \mid [\tilde{y} \mapsto \tilde{v}]P_2$. By the typing rules, it must be the case that:

$$\begin{array}{l}\Gamma_1, x : \langle\tilde{\tau}\rangle^{l_1}/U_1 \vdash_l P_1 \\ \Gamma_2, x : \langle\tilde{\tau}\rangle^{l_2}/U_2, \tilde{y} : \tilde{\tau} \vdash_l P_2 \\ \tilde{\tau'} \leq \uparrow\tilde{\tau} \\ \Gamma \leq (\Gamma_1 \mid \tilde{v} : \tilde{\tau'} \mid x : \langle\tilde{\tau}\rangle^{l_1}/O^0_{t_1}.U_1) \mid (\Gamma_2, x : \langle\tilde{\tau}\rangle^{l_2}/I^0_{t_2}.U_2) \\ l \sqsubseteq l_1, l_2\end{array}$$

  By the substitution lemma (Lemma C.8), we have:

$$(\Gamma_2, x : \langle\tilde{\tau}\rangle^{l_2}/U_2) \mid \tilde{v} : \tilde{\tau} \vdash_l P_2$$

  Let $\Delta' = (\Gamma_1 \mid \tilde{v} : \tilde{\tau} \mid x : \langle\tilde{\tau}\rangle^{l_1}/U_1) \mid (\Gamma_2, x : \langle\tilde{\tau}\rangle^{l_2}/U_2)$. Then, $\Delta' \vdash_l Q$ holds. Moreover, since $\tilde{\tau'} \leq \uparrow\tilde{\tau} \leq \tilde{\tau}$ (note that $\uparrow\tau \leq \tau$ follows from the sixth law of Lemma C.1), we have $\Gamma \leq (\Gamma_1 \mid \tilde{v} : \tilde{\tau} \mid x : \langle\tilde{\tau}\rangle^{l_1}/O^0_{t_1}.U_1) \mid (\Gamma_2, x : \langle\tilde{\tau}\rangle^{l_2}/I^0_{t_2}.U_2) \longrightarrow \Delta'$. By Lemma C.2, there exists Δ such that $\Gamma \longrightarrow \Delta$ and $\Delta \leq \Delta'$. So, we have $\Delta \vdash_l Q$ and $\Gamma \longrightarrow \Delta$ as required.

- Case for R-Par: In this case, $P = P_1 \mid P_2$ and $Q = Q_1 \mid P_2$ with $P_1 \longrightarrow Q_1$. By the typing rules, there exist Γ_1 and Γ_2 such that $\Gamma_i \vdash_l P_i$ and $\Gamma \leq \Gamma_1 \mid \Gamma_2$. By the induction hypothesis, there exists Δ_1 such that $\Gamma_1 \longrightarrow \Delta_1$ or $\Delta_1 = \Gamma_1$. Let $\Delta' = \Delta_1 \mid \Gamma_2$. Then, $\Delta' \vdash_l Q$, and either $\Gamma_1 \mid \Gamma_2 \longrightarrow \Delta'$ or $\Delta' = \Gamma_1 \mid \Gamma_2$ holds. In the latter case, the required result holds for $\Delta = \Gamma$. In the former case, by Lemma C.2, there exists Δ such that $\Gamma \longrightarrow \Delta$ and $\Delta \leq \Delta'$. By using T-Sub, we obtain $\Delta \vdash_l Q$ as required.

- Case for R-New: In this case, $P = (\nu x : \xi)\, P_1$ and $Q = (\nu x : \xi)\, Q_1$ with $P_1 \longrightarrow Q_1$. By the typing rules, $\Gamma, x : \xi/U \vdash_l P_1$ with $rel(U)$. By the induction hypothesis, either $\Gamma, x : \xi/U \vdash_l Q_1$ holds, or there exists Δ and U' such that $\Delta, x : \xi/U' \vdash_l Q_1$ and $\Gamma, x : \xi/U \longrightarrow \Delta, x : \xi/U'$. In the former case, $\Delta = \Gamma$ satisfies the required condition. In the latter case, by Lemma C.3, $rel(U')$ holds, so that we can obtain $\Delta \vdash_l Q$ as required.

- Case for R-SP: This follows immediately from Lemma C.6 and the induction hypothesis.

□ □

D Proof of Theorem 4.10

D.1 Basic Properties about the Operational Semantics

Lemma D.1: If $P \longrightarrow Q$, then $P \preceq (\nu\tilde{u})\,(\overline{x}\langle\tilde{v}\rangle.\, P_1 \mid x(\tilde{y}).\, P_2 \mid P_3)$ and $(\nu\tilde{u})\,(P_1 \mid [\tilde{y} \mapsto \tilde{v}]P_2 \mid P_3) \preceq Q$ for some $\tilde{u}, x, \tilde{v}, P_1, P_2, P_3$.

Proof: This follows by straightforward induction on derivation of $P \longrightarrow Q$. □

D.2 Basic Properties of the Erasure Function

Lemma D.2: If $\Gamma \sim \Delta$, then $\mathbf{Er}_\Gamma(P) = \mathbf{Er}_\Delta(P)$.

Proof: Straightforward induction on the structure of P. □

Lemma D.3: If $\Gamma(v) \sim \tau$, then $\mathbf{ErV}_\Gamma([x \mapsto v]v') = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma, x : \tau}(v')$.

Proof: If $x \neq v'$, then $\mathbf{ErV}_\Gamma([x \mapsto v]v') = \mathbf{ErV}_\Gamma(v') = \mathbf{ErV}_{\Gamma,x:\tau}(v') = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(v')$. Suppose that $x = v'$. If $\mathbf{High}(\tau)$, then the result follows from $\mathbf{ErV}_\Gamma(v) = \mathbf{ErV}_{\Gamma,x:\tau}(x) = \star$. Otherwise, $\mathbf{ErV}_{\Gamma,x:\tau}(x) = x$. So, we have $\mathbf{ErV}_\Gamma([x \mapsto v]v') = \mathbf{ErV}_\Gamma(v) = [x \mapsto \mathbf{ErV}_\Gamma(v)]x = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(x)$. □

Lemma D.4: If $\Gamma(v) \sim \tau$, then $\mathbf{Er}_\Gamma([x \mapsto v]P) = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(P)$.

Proof: This follows by straightforward induction on the structure of P. We show only the case for output processes. The other cases are similar or trivial.

- Case where P is of the form $\overline{y}\langle\tilde{w}\rangle.\,Q$.
  By the induction hypothesis, $\mathbf{Er}_\Gamma([x \mapsto v]Q) = [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(Q)$. Note that $\Gamma([x \mapsto v]y) \sim \langle\tilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$ if and only if $(\Gamma, x:\tau)(y) \sim \langle\tilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$. So, if $\Gamma([x \mapsto v]y) \not\sim \langle\tilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$, then the result follows from the following equations.
  $$\begin{array}{ll} & \mathbf{Er}_\Gamma([x \mapsto v]P) \\ = & \mathbf{Er}_\Gamma([x \mapsto v]Q) \\ = & [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(Q) \\ = & [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(P) \end{array}$$
  Suppose that $\Gamma([x \mapsto v]y) \sim \langle\tilde{\sigma}\rangle^{\mathbf{L}}/\mathbf{0}$. Let $y' = [x \mapsto v]y$. Then, $y' = \mathbf{Er}_\Gamma(y') = [x \mapsto \mathbf{Er}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(y) = [x \mapsto \mathbf{Er}_\Gamma(v)]y$. (The second equality follows from Lemma D.3.) Thus, the result follows from the following equations.
  $$\begin{array}{ll} & \mathbf{Er}_\Gamma([x \mapsto v]P) \\ = & \overline{y'}\langle\mathbf{ErV}_\Gamma([x \mapsto v]\tilde{w})\rangle.\,\mathbf{Er}_\Gamma([x \mapsto v]Q) \\ = & \overline{y'}\langle[x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{ErV}_{\Gamma,x:\tau}(\tilde{w})\rangle. \\ & \quad [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(Q) \\ = & [x \mapsto \mathbf{ErV}_\Gamma(v)]\mathbf{Er}_{\Gamma,x:\tau}(P) \end{array}$$

□

Lemma D.5: If $\mathbf{High}(\Gamma(x))$, then $x \notin \mathbf{Er}_\Gamma(P)$.

Proof: Straightforward induction on the structure of P. □

D.3 Lock-freedom Property

In this subsection, we show that input/output actions whose capability levels are finite succeed eventually.

We define x^α by: $x^I = x$ and $x^O = \overline{x}$.

Lemma D.6: Suppose that the following conditions hold.

1. $\Gamma \vdash_l P$
2. $\Delta \vdash_l Q$
3. $ob_\alpha(\Gamma(x)) \in \mathbf{Nat}$
4. $closed(\Gamma \mid \Delta)$

Then there exists R such that $P \mid Q \longrightarrow_{\mathbf{H}}^* R$ and $x^\alpha \in SBarbs(R)$.

Proof: Let n be $ob_O(\Gamma(x))$. The proof proceeds by well-founded induction on $(n, \#(P))$, where the well-founded order is defined by $(n, m) < (n', m') \iff (n < n') \vee (n = n' \wedge m < m')$. We perform case analysis on P. We will consider only the case for $\alpha = O$ below: The case for $\alpha = I$ is similar.

- Case $P = \mathbf{0}$: This case cannot happen.

- Case $P = \overline{y}\langle\tilde{v}\rangle^{l_1,t}.\,P_1$. If $y = x$, then the result follows immediately. If $y \neq x$, then by the rule T-Out, it must be the case that:

$$\begin{array}{l}\Gamma_1, y : \langle\tilde{\tau}\rangle^{l_1}/U \vdash_{l'} P_1\\ t < ob(\tilde{v} : \tilde{\tau'} \mid \Gamma_1)\\ \tilde{\tau'} \leq \uparrow\tilde{\tau}\\ \Gamma \leq \Gamma_1 \mid \tilde{v} : \tilde{\tau'}, y : \langle\tilde{\tau}\rangle^{l_1}/O^0_t.U\end{array}$$

  The second and fourth conditions imply $t < ob_O((\Gamma_1 \mid \tilde{v} : \tilde{\tau'})(x)) < ob_O(\Gamma(x)) = n$ (which also implies $l_1 = \mathbf{H}$ by the well-formedness condition of types). Since $closed(\Gamma \mid \Delta)$ holds, Lemma C.4 implies $closed((\Gamma_1 \mid \tilde{v} : \tilde{\tau'}, y : \langle\tilde{\tau}\rangle^{l_1}/O^0_t.U) \mid \Delta)$, from which we obtain $rel(\langle\tilde{\tau}\rangle^{l_1}/O^0_t.U \mid \Delta(y))$. So, it must be the case that $ob_I(\Delta)(y) \leq t < n$. By the induction hypothesis, it must be the case that $Q \longrightarrow^*_{\mathbf{H}} \preceq (\nu\tilde{u})\,(y(\tilde{z}).\,Q_1 \mid Q_2)$. By Lemma C.6 and Theorem 4.1, $\Delta' \vdash_l (\nu\tilde{u})\,(y(\tilde{z}).\,Q_1 \mid Q_2)$ holds for some Δ' such that $\Delta \longrightarrow^* \Delta'$. By the typing rules, it must be the case that

$$\begin{array}{l}\Delta_1, y : \langle\tilde{\tau}\rangle^{\mathbf{H}}/V, \tilde{z} : \tilde{\tau} \vdash_{l_1} Q_1\\ \Delta_2 \vdash_l Q_2\\ \Delta', \tilde{u} : \tilde{\sigma} \leq (\Delta_1, y : \langle\tilde{\tau}\rangle^{\mathbf{H}}/I^{t_1}_{t_2}.V) \mid \Delta_2\end{array}$$

  By the substitution lemma (Lemma C.8, $(\Delta_1, y : \langle\tilde{\tau}\rangle^{\mathbf{H}}/V) \mid \tilde{v} : \tilde{\tau} \vdash_{l_1} [\tilde{z} \mapsto \tilde{v}]Q_1$. So, we have:

$$(\Gamma_1 \mid \tilde{v} : \tilde{\tau}, y : \langle\tilde{\tau}\rangle^{\mathbf{H}}/U \mid V) \mid \Delta_1 \mid \Delta_2 \vdash_l P_1 \mid [\tilde{z} \mapsto \tilde{v}]Q_1 \mid Q_2$$

  Moreover, $closed(\Gamma \mid \Delta)$ and Lemma C.3 imply $closed((\Gamma_1, y : \langle\tilde{\tau}\rangle^{\mathbf{H}}/(U \mid V)) \mid \Delta_1 \mid \Delta_2)$. The condition $\Gamma \leq \Gamma_1 \mid \tilde{v} : \tilde{\tau'}, y : \langle\tilde{\tau}\rangle^{l_1}/O^0_t.U$ implies $ob_O(\Gamma_1 \mid \tilde{v} : \tilde{\tau'})(x) \leq n$, which implies either (i)$ob_O(\Gamma_1(x)) \leq n$ or (ii)$ob_O((\tilde{v} : \tilde{\tau'})(x)) \leq n$. In the former case, since $\#P_1 < \#P$ holds, the induction hypothesis implies that there must exist R such that $P_1 \mid [\tilde{z} \mapsto \tilde{v}]Q_1 \mid Q_2 \longrightarrow^*_{\mathbf{H}} R$ and $\overline{x} \in SBarbs(R)$. In the latter case, $ob_O((\tilde{v} : \tilde{\tau})(x)) < ob_O((\tilde{v} : \tilde{\tau'})(x)) \leq n$. So, by the induction hypothesis, $P_1 \mid [\tilde{z} \mapsto \tilde{v}]Q_1 \mid Q_2 \longrightarrow_{\mathbf{H}} * R$ with $\overline{x} \in SBarbs(R)$. The required result follows, since $P \mid Q \longrightarrow^*_{\mathbf{H}} (\nu\tilde{u})\,(P_1 \mid [\tilde{z} \mapsto \tilde{v}]Q_1 \mid Q_2)$.

- Case for $P = y(\tilde{z}).\,P_1$: Similar to the above case.

- Case for $P = P_1 \mid P_2$: There exist Γ_1 and Γ_2 such that $\Gamma \leq \Gamma_1 \mid \Gamma_2$ and $\Gamma_i \vdash_l P_i$. By Lemma C.4, $closed(\Gamma_1 \mid \Gamma_2 \mid \Delta)$ holds. Since $ob_O(\Gamma(x)) = n$, either $ob_O(\Gamma_1(x)) \leq n$ or $ob_O(\Gamma_2(x)) \leq n$ holds. In the former case, since $\#(P_1) < \#(P)$ holds, we can apply induction hypothesis to obtain $P \mid Q \preceq P_1 \mid (P_2 \mid Q) \longrightarrow_{\mathbf{H}} * R$ with $\overline{x} \in SBarbs(R)$. The latter case is similar.

- Case for $P = *P_1$: There exists Γ_1 such that $\Gamma_1 \vdash_l P_1$ and $\Gamma \leq *\Gamma_1$. Since $\Gamma \leq \Gamma_1 \mid \Gamma$, $closed(\Gamma_1 \mid (\Gamma \mid \Delta))$ holds. So, by applying the induction hypothesis to P_1, we obtain R such that $P_1 \mid (P \mid Q) \longrightarrow^*_{\mathbf{H}} R$ with $\overline{x} \in SBarbs(R)$. The required result follows, since $P \mid Q \preceq P_1 \mid (P \mid Q)$.

- Case for $P = (\nu y : \xi)\,P_1$: By the typing rules, we have $\Gamma, y : \xi/U \vdash_l P_1$. Since $closed((\Gamma, y : \xi/U) \mid \Delta$ and $\#(P_1) < \#(P)$ hold, we can apply induction hypothesis to obtain R' such that $P_1 \mid Q \longrightarrow^*_{\mathbf{H}} R'$ and $\overline{x} \in SBarbs(R')$. Thus, the required result holds for $R = (\nu y : \xi)\,R'$.

- Case for $P = \mathbf{if}\ b\ \mathbf{then}\ P_1\ \mathbf{else}\ P_2$: By the assumption $closed(\Gamma)$, b is either $true^{l'}$ or $false^{l'}$. By the typing rules, $\Gamma \vdash_l P_i$ holds. So, by the induction hypothesis, there exists R such that $P_i \mid Q \longrightarrow^*_{\mathbf{H}} R$ and $\overline{x} \in SBarbs(R)$. The required result follows, since $P \preceq P_i$ for $i = 1$ or 2.

□

Theorem D.7: Let C be a finite-level context. If $\Gamma \vdash_l C[P]$ and $\mathit{closed}(\Gamma)$, then $C \longrightarrow^*_{\mathbf{H}} E$ for some evaluation context E.

Proof: The proof proceeds by induction on the depth of the hole in C If the depth is 0, the result follows immediately (since C is itself an evaluation context). If C is of the form $E_1[\overline{x}\langle\tilde{v}\rangle.\, C_1]$, then by Lemma D.6, there exists R such that $E[\overline{x}\langle\tilde{v}\rangle.\, C_1[P]] \longrightarrow^*_{\mathbf{H}} R$ with $x \in \mathit{Barbs}(R)$. So, $E_1[\overline{x}\langle\tilde{v}\rangle.\, C_1] \longrightarrow^*_{\mathbf{H}} E_1'[C_1]$ for some evaluation context E_1'. By the induction hypothesis, there exists E such that $E_1'[C_1] \longrightarrow^*_{\mathbf{H}} E$. Therefore, we have $C \longrightarrow^*_{\mathbf{H}} E$ as required. The case where C is of the form $E_1[x(\tilde{y}).\, C_1]$ is similar. □

D.4 Simulation of P by $\mathbf{Er}_\Gamma(P)$

In this section, we show that the behavior of P can be simulated by its erasure $\mathbf{Er}_\Gamma(P)$.

Lemma D.8: If $P \preceq Q$, then $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(Q)$.

Proof: This follows by straightforward induction on derivation of $P \preceq Q$. The only non-trivial is the case where S-IfT or S-IfF is applied.

- Case S-IfT: In this case, $P = \mathbf{if}\ \mathit{true}^{l'}\ \mathbf{then}\ Q\ \mathbf{else}\ R$. If $l' = \mathbf{L}$, then $\mathbf{Er}_\Gamma(P) = \mathbf{if}\ \mathit{true}^{l'}\ \mathbf{then}\ \mathbf{Er}_\Gamma(Q)\ \mathbf{else}\ \mathbf{Er}_\Gamma(R)$, so that we have $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(Q)$ as required. If $l' = \mathbf{H}$, then $\mathbf{Er}_\Gamma(P) = \mathbf{0}$. By the assumption $\Gamma \vdash_l \mathbf{if}\ \mathit{true}^{\mathbf{H}}\ \mathbf{then}\ Q\ \mathbf{else}\ R$, it must be the case that $\Gamma \vdash_{\mathbf{H}} Q$. So, by Lemma 4.9, we have $\mathbf{Er}_\Gamma(P) = \mathbf{0} \preceq \mathbf{Er}_\Gamma(Q)$ as required.
- Case S-IfF: Similar to the case for R-IfT.

□

Lemma D.9: Suppose $\Gamma \vdash_l P$. If $P \longrightarrow_{\mathbf{H}} Q$, then $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(Q)$. If $P \longrightarrow_{\mathbf{L}} Q$, $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(Q)$.

Proof: This follows by induction on derivation for $P \longrightarrow_{l_1} Q$ with case analysis on the last rule used.

- Case R-Com: In this case, $P = \overline{x}\langle\tilde{v}\rangle^{l_1,t_1}.\, P_1 \mid x(\tilde{y})^{l_1,t_2}.\, P_2$ and $Q = P_1 \mid [\tilde{y} \mapsto \tilde{v}]P_2$. By the assumption $\Gamma \vdash_l P$, $\Gamma(x)$ is of the form $\langle\tilde{\tau}\rangle^{l_1}/U$. If $l_1 = \mathbf{L}$, then $\mathbf{Er}_\Gamma(P) = \overline{x}\langle\tilde{v'}\rangle.\, \mathbf{Er}_\Gamma(P_1) \mid x(\tilde{y}).\, \mathbf{Er}_{\Gamma,\tilde{y}:\tilde{\tau}}(P_2)$ where $v_i' = \mathbf{ErV}_{\tau_i}(v_i)$. By Lemma D.4, $\mathbf{Er}_\Gamma([\tilde{y} \mapsto \tilde{v}]P_2) = [\tilde{y} \mapsto \tilde{v'}]\mathbf{Er}_{\Gamma,\tilde{y}:\tilde{\tau}}(P_2)$. So, we have $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(Q)$ as required.

  If $l_1 = \mathbf{H}$, then $\mathbf{Er}_\Gamma(P) = \mathbf{Er}_\Gamma(P_1) \mid \mathbf{Er}_{\Gamma,\tilde{y}:\tilde{\tau}}(P_2)$. $\mathbf{Er}_\Gamma(Q) = \mathbf{Er}_\Gamma(P_1) \mid \mathbf{Er}_\Gamma([\tilde{y} \mapsto \tilde{v}]P_2)$. By the condition on well-formed types, $\mathbf{High}(\tau_i)$ holds. By Lemmas D.4 and D.5, we have:

$$\begin{array}{ll} & \mathbf{Er}_\Gamma([\tilde{y} \mapsto \tilde{v}]P_2) \\ = & [\tilde{y} \mapsto \mathbf{Er}_\Gamma(\tilde{v})]\mathbf{Er}_{\Gamma,\tilde{y}:\tilde{\tau}}(P_2) \\ = & \mathbf{Er}_{\Gamma,\tilde{y}:\tilde{\tau}}(P_2) \end{array}$$

  So, $\mathbf{Er}_\Gamma(P) = \mathbf{Er}_\Gamma(Q)$ holds.
- Case R-Par: In this case, $P = P_1 \mid P_2$ and $Q = Q_1 \mid P_2$ with $P_1 \longrightarrow_{l_1} Q_1$. By the assumption $\Gamma \vdash_l P$, there must exist Γ_1 such that $\Gamma_1 \vdash_l P_1$ and $\Gamma \sim \Gamma_1$. By the induction hypothesis, $\mathbf{Er}_{\Gamma_1}(P_1) \longrightarrow \mathbf{Er}_{\Gamma_1}(Q_1)$ holds if $l_1 = \mathbf{L}$ and $\mathbf{Er}_{\Gamma_1}(P_1) \preceq \mathbf{Er}_{\Gamma_1}(Q_1)$ holds if $l_1 = \mathbf{H}$. So, $\mathbf{Er}_{\Gamma_1}(P) \longrightarrow \mathbf{Er}_{\Gamma_1}(Q)$ holds if $l_1 = \mathbf{L}$ and $\mathbf{Er}_{\Gamma_1}(P) \equiv_0 \mathbf{Er}_{\Gamma_1}(Q)$ holds if $l_1 = \mathbf{H}$. Since $\Gamma \sim \Gamma_1$, Lemma D.2 implies $\mathbf{Er}_{\Gamma_1}(P) = \mathbf{Er}_\Gamma(P)$ and $\mathbf{Er}_{\Gamma_1}(Q) = \mathbf{Er}_\Gamma(Q)$. Thus, we have the required result.
- Case R-New: Trivial by the induction hypothesis.

- Case R-SPCONG: In this case, $P \preceq P'$, $P' \longrightarrow_{l_1} Q'$, and $Q' \preceq Q$. By Lemma D.8, $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(P')$ and $\mathbf{Er}_\Gamma(Q') \preceq \mathbf{Er}_\Gamma(Q)$ hold. By Lemma C.6, $\Gamma \vdash_l P'$. So, by the induction hypothesis, $\mathbf{Er}_\Gamma(P') \longrightarrow \mathbf{Er}_\Gamma(Q')$ holds if $l_1 = \mathbf{L}$, and $\mathbf{Er}_\Gamma(P') \equiv_0 \mathbf{Er}_\Gamma(Q')$ holds if $l_1 = \mathbf{H}$. Therefore, $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(Q)$ holds if $l_1 = \mathbf{L}$ and $\mathbf{Er}_\Gamma(P) \equiv_0 \mathbf{Er}_\Gamma(Q)$ holds if $l_1 = \mathbf{H}$.

□

Lemma D.10: If Γ is a low-level type environment and $\Gamma \vdash_l P$, then $\mathit{SBarbs}(P) \subseteq \mathit{SBarbs}(\mathbf{Er}_\Gamma(P))$.

Proof: Suppose that $\overline{x} \in \mathit{SBarbs}(P)$. Then, $P \preceq (\nu\tilde{y})\,(\overline{x}\langle\tilde{v}\rangle.\,Q \mid R)$ with $x \notin \{\tilde{y}\}$. By Lemma D.8, $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma((\nu\tilde{y})\,(\overline{x}\langle\tilde{v}\rangle.\,Q \mid R))$ By the assumptions that Γ is a low-level type environment and that $\Gamma \vdash_l P$, $\Gamma(x)$ is of the form $\langle\tilde{\tau}\rangle^{\mathbf{L}}/U$. So, $\mathbf{Er}_\Gamma((\nu\tilde{y})\,(\overline{x}\langle\tilde{v}\rangle.\,Q \mid R))$ is of the form $(\nu\tilde{y'})\,(\overline{x}\langle\tilde{v'}\rangle.\,Q' \mid R')$. Thus $\overline{x} \in \mathit{SBarbs}(\mathbf{Er}_\Gamma(P))$ holds as required. Similarly, $x \in \mathit{SBarbs}(P)$ implies $x \in \mathit{SBarbs}(\mathbf{Er}_\Gamma(P))$. □

D.5 Simulation of $\mathbf{Er}_\Gamma(P)$ by P

In this subsection, we show that the behavior of $\mathbf{Er}_\Gamma(P)$ can be simulated by the original process P.

Lemma D.11: Suppose that $\mathit{closed}(\Gamma)$ and $\Gamma \vdash_l Q$ hold. If $P_1 \preceq P_1'$ and $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$, then there exists Q' such that $Q \longrightarrow^* \preceq Q'$ and $P_1' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q')$.

Proof: The proof proceeds by induction on derivation of $P_1 \preceq P_1'$ with case analysis on the last rule used.

- Case for the rule for reflexivity: Since $P_1' = P_1$, the result follows for $Q' = Q$.
- Case for the rule for transitivity: In this case, $P_1 \preceq P_1'' \preceq P_1'$. By the induction hypothesis, there exists Q'' such that $Q \longrightarrow^* \preceq Q''$ and $P_1'' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q'')$. By Theorem 4.1 and Lemmas C.3 and C.6, there exists Γ' such that $\Gamma' \vdash_l Q''$ and $\mathit{closed}(\Gamma')$ holds. By the induction hypothesis, there exists Q' such that $Q'' \longrightarrow^* \preceq Q'$ and $P_1' \mid P_2 \preceq \mathbf{Er}_{\Gamma'}(Q')$. By Lemma D.2, we have $P_1' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q')$ as required.
- Cases for S-Zero1, S-Zero2, S-Zero3, S-Commut, S-Assoc, S-New, and S-Swap: Trivial. (Let $Q' = Q$.)
- Case for S-IfT: In this case, $P_1 = \mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ P_1'\ \mathbf{else}\ R$. If $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$ has been derived from $P_1 \preceq P_1'$, then $Q' = Q$ satisfies the required condition. Otherwise, $\mathbf{Er}_\Gamma(Q) = E[P_1]$ and $[\,]_{FV(P_1)} \mid P_2 \preceq E$ for some evaluation context E. So, by the definition of $\mathbf{Er}$, $Q = C[\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2]$ for some finite-level context C and Q_1 such that $\mathbf{Er}_\Gamma(C) = E$, $\mathbf{Er}_\Delta(Q_1) = P_1'$, and $\mathbf{Er}_\Delta(Q_2) = R$ where $\Delta = \mathit{ext}(\Gamma, C)$. By Theorem D.7, $C \longrightarrow_{\mathbf{H}}^* E'$ for some evaluation context E'. Let $Q' = E'[Q_1]$. Then,

$$\begin{array}{rcl} Q & = & C[\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2] \\ & \longrightarrow_{\mathbf{H}}^* & E'[\mathbf{if}\ \mathit{true}^{\mathbf{L}}\ \mathbf{then}\ Q_1\ \mathbf{else}\ Q_2] \\ & \preceq & E'[Q_1] \\ & = & Q' \end{array}$$

  Moreover, by Lemma D.9, $\mathbf{Er}_\Gamma(C[Q_1]) \preceq \mathbf{Er}_\Gamma(E'[Q_1]) = \mathbf{Er}_\Gamma(Q')$ holds, which implies $P_1' \mid P_2 \preceq E[P_1'] = \mathbf{Er}_\Gamma(C[Q_1]) \preceq \mathbf{Er}_\Gamma(Q')$.
- Case for S-IfF: Similar to the case for S-IfT.

- Case for S-Rep: In this case, $P_1 = *P_{11}$ and $P_1' = *P_{11} \mid P_{11}$. If $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$ has been derived from $*P_{11} \preceq *P_{11} \mid P_{11}$, then $Q' = Q$ satisfies the required condition. Otherwise, $\mathbf{Er}_\Gamma(Q) = E[P_1]$ and $[\,]_{FV(P_1)} \mid P_2 \preceq E$ for some evaluation context E. So, $Q = C[*Q_1]$ for some context context C and process Q_1, such that $\mathbf{Er}_\Gamma(C) = E$ and $\mathbf{Er}_\Delta(Q_1) = P_{11}$ for $\Delta = ext(\Gamma, C)$. If $P_{11} \equiv \mathbf{0}$, then $Q' = Q$ satisfies the required condition. Otherwise, by the definition of $\mathbf{Er}$, C is a finite-level context. By Theorem D.7, $C \longrightarrow_{\mathbf{H}}^* E'$ holds for some evaluation context E'. Let $Q' = E'[*Q_1 \mid Q_1]$. Then, $Q = C[*Q_1] \longrightarrow^* \preceq E'[*Q_1] \preceq Q'$. Moreover, by Lemma D.9, $\mathbf{Er}_\Gamma(C[*Q_1 \mid Q_1]) \preceq \mathbf{Er}_\Gamma(E'[*Q_1 \mid Q_1]) = \mathbf{Er}_\Gamma(Q')$ holds, which implies $P_1' \mid P_2 \preceq E[P_1'] = \mathbf{Er}_\Gamma(C[*Q_1 \mid Q_1]) \preceq \mathbf{Er}_\Gamma(Q')$.

- Case for S-Par: In this case, $P_1 = P_{11} \mid P_{12}$ and $P_1' = P_{11}' \mid P_{12}$ with $P_{11} \preceq P_{11}'$. By the assumption $P_1 \mid P_2 \preceq \mathbf{Er}_\Gamma(Q)$, we have $P_{11} \mid (P_{12} \mid P_2) \preceq \mathbf{Er}_\Gamma(Q)$. By the induction hypothesis, there exists Q' such that $Q \longrightarrow^* \preceq Q'$ and $P_{11}' \mid (P_{12} \mid P_2) \preceq \mathbf{Er}_\Gamma(Q')$. The required result follows, since $P_{11}' \mid P_2 \preceq P_{11}' \mid (P_{12} \mid P_2)$.

- Case for S-CNew: In this case, $P_1 = (\nu x : \xi)\, P_{11}$ and $P_1' = (\nu x : \xi)\, P_{11}'$ with $P_{11} \preceq P_{11}'$. Let Q_1 be a process obtained by removing the prefix $(\nu x : \xi)$ from Q. Then, we have $P_{11} \mid P_2 \preceq \mathbf{Er}_{\Gamma, x : \xi/\mathbf{0}}(Q_1)$. By the induction hypothesis, there exists Q_1' such that $Q_1 \longrightarrow^* \preceq Q_1'$ and $P_{11}' \mid P_2 \preceq \mathbf{Er}_{\Gamma, x : \xi/\mathbf{0}}(Q_1')$. Let $Q' = (\nu x : \xi)\, Q_1$. Then, we have $Q \longrightarrow^* \preceq Q'$ and $P_1' \mid P_2 \preceq \mathbf{Er}_\Gamma(Q')$ as required.

□

Corollary D.12: Suppose *closed*(Γ) and $\Gamma \vdash_l Q$. If $P \preceq \mathbf{Er}_\Gamma(Q)$ and $P \preceq P'$, then there exists Q' such that $Q \longrightarrow^* Q'$ and $P' \preceq \mathbf{Er}_\Gamma(Q')$.

Proof: Let $P_1 = P$, $P_1' = P'$, and $P_2 = \mathbf{0}$ in Lemma D.11. Then, there exists Q' such that $Q \longrightarrow^* \preceq Q'$ and $P' \mid \mathbf{0} \preceq \mathbf{Er}_\Gamma(Q')$. From the second condition and S-Zero1, $P' \preceq \mathbf{Er}_\Gamma(Q')$ follows. □

We write $\longrightarrow^+$ for the transitive closure of $\longrightarrow^*$.

Lemma D.13: Suppose *closed*(Γ) and $\Gamma \vdash_l P$. If $P' \preceq \mathbf{Er}_\Gamma(P)$ and $P' \longrightarrow Q'$, then there exists Q such that $P \longrightarrow^+ Q$ and $Q' \preceq \mathbf{Er}_\Gamma(Q)$.

Proof: By Lemma D.1, we have $P' \preceq (\nu \tilde{u})\,(\overline{x}\langle \tilde{v} \rangle.\, P_1' \mid x(\tilde{y}).\, P_2' \mid P_3')$ and $(\nu \tilde{u})\,(P_1' \mid [\tilde{y} \mapsto \tilde{v}]P_2' \mid P_3') \preceq Q'$ for some P_1', P_2', P_3'. By Corollary D.12, there exists R such that $P \longrightarrow^* \preceq R$ and $(\nu \tilde{u})\,(\overline{x}\langle \tilde{v'} \rangle.\, P_1' \mid x(\tilde{y}).\, P_2' \mid P_3') \preceq \mathbf{Er}_\Gamma(R)$. So, $\mathbf{Er}_\Gamma(R)$ is of the form $E[\overline{x}\langle \tilde{v'} \rangle.\, P_1', x(\tilde{y}).\, P_2']$ with $(\nu \tilde{u})\,([\,]^{(1)}_{id, S_1} \mid [\,]^{(2)}_{id, S_2} \mid P_3') \preceq E$ where $S_1 = FV(\overline{x}\langle \tilde{v'} \rangle.\, P_1')$ and $S_2 = FV(x(\tilde{y}).\, P_2')$. By the definition of $\mathbf{Er}$, $R = C[\overline{x}\langle \tilde{v} \rangle.\, P_1, x(\tilde{y}).\, P_2]$ for a finite-level context C with two holes such that $\mathbf{Er}_\Gamma(C) = E$ with $\mathbf{Er}_{\Delta_1}(\overline{x}\langle \tilde{v} \rangle.\, P_1) = \overline{x}\langle \tilde{v'} \rangle.\, P_1'$ and $\mathbf{Er}_{\Delta_2}(x(\tilde{y}).\, P_2) = x(\tilde{y}).\, P_2'$ where $\Delta_i = ext^{(i)}(\Gamma, C)$. By Lemma D.7, there exists an evaluation context E' with two holes such that $C \longrightarrow^* E'$. Let $Q'' = E'[P_1, [\tilde{y} \mapsto \tilde{v}]P_2]$. Then $R \longrightarrow^+ Q''$ holds. Moreover, $(\nu \tilde{u})\,(P_1' \mid [\tilde{y} \mapsto \tilde{v'}]P_2' \mid P_3') \preceq E[P_1', [\tilde{y} \mapsto \tilde{v'}]P_2'] = \mathbf{Er}_\Gamma(C[P_1, [\tilde{y} \mapsto \tilde{v}]P_2]) \preceq \mathbf{Er}_\Gamma(E'[P_1, [\tilde{y} \mapsto \tilde{v}]P_2]) = \mathbf{Er}_\Gamma(Q'')$. By Corollary D.12 and $(\nu \tilde{u})\,(P_1' \mid [\tilde{y} \mapsto \tilde{v}]P_2' \mid P_3') \preceq Q'$, there exists Q such that $Q' \preceq \mathbf{Er}_\Gamma(Q)$ and $Q'' \longrightarrow^* \preceq Q$. Moreover, we have $P \longrightarrow^* \preceq R \longrightarrow^+ Q'' \longrightarrow^* \preceq Q$, which implies $P \longrightarrow^+ Q$. □

Lemma D.14: If *closed*(Γ) and $\Gamma \vdash_l P$, then $SBarbs(\mathbf{Er}_\Gamma(P)) \subseteq Barbs(P)$.

Proof: Suppose $\overline{x} \in SBarbs(\mathbf{Er}_\Gamma(P))$. Then, $\mathbf{Er}_\Gamma(P) = E[\overline{x}\langle \tilde{v'} \rangle P_1']$ for some evaluation context E. By the definition of $\mathbf{Er}$, $P = C[\overline{x}\langle \tilde{v} \rangle P_1]$ for some finite-level context C and process P_1. By Theorem D.7, $C \longrightarrow^* E'$ for some finite-level context. So, $P \longrightarrow^* E'[\overline{x}\langle \tilde{v} \rangle P_1]$, which implies $\overline{x} \in Barbs(P)$. □

D.6 Proof of Theorem 4.10

Proof of Theorem 4.10: Let $\mathcal{R} = \{(P, Q) \mid \Gamma \vdash_l P, Q \preceq \mathbf{Er}_\Gamma(P),\ \Gamma \text{ is low-level and closed}\}$. We show that $\mathcal{R}$ is a barbed bisimulation. Suppose $(P, Q) \in \mathcal{R}$, i.e., $\Gamma \vdash_l P$ and $Q \preceq \mathbf{Er}_\Gamma(P)$ for some low-level, closed Γ. We check the three conditions of Definition 4.3.

- If $P \longrightarrow P'$, by Theorem 4.1, there exists Δ such that $\Gamma \longrightarrow^* \Delta$ and $\Delta \vdash_l P'$. Moreover, by Lemma D.9, either $\mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(P')$ or $\mathbf{Er}_\Gamma(P) \longrightarrow \mathbf{Er}_\Gamma(P')$ holds. In the former case, let Q' be Q. Then, $Q' \preceq \mathbf{Er}_\Gamma(P) \preceq \mathbf{Er}_\Gamma(P') = \mathbf{Er}_\Delta(P')$. In the latter case, let Q' be $\mathbf{Er}_\Gamma(P')(= \mathbf{Er}_\Delta(P'))$. $Q \longrightarrow^* Q'$ and $(P', Q') \in \mathcal{R}$ hold in either case.

- If $Q \longrightarrow Q'$, by Lemma D.13, there exists P' such that $P \longrightarrow^+ P'$ and $Q' \preceq \mathbf{Er}_\Gamma(P')$. By Theorem 4.1, there exists Δ such that $\Gamma \longrightarrow^* \Delta$ and $\Delta \vdash_l P'$. By Lemma D.2, $\mathbf{Er}_\Gamma(P') = \mathbf{Er}_\Delta(P')$. Therefore, we have $(P', Q') \in \mathcal{R}$ as required.

- Suppose that $\chi \in \mathit{Barbs}(P)$. Then there exists P' such that $P \longrightarrow^* P'$ and $\chi \in \mathit{SBarbs}(P')$. By the first condition of barbed bisimulation, there exists Q' such that $Q \longrightarrow^* Q'$ and $Q' \preceq \mathbf{Er}_\Gamma(P')$. By Lemma D.10, $\chi \in \mathit{Barbs}(\mathbf{Er}_\Gamma(P')) = \mathit{SBarbs}(Q')$. So, $\chi \in \mathit{Barbs}(Q)$ holds.

  On the other hand, suppose that $\chi \in \mathit{Barbs}(Q)$ holds. Then there exists Q' such that $Q \longrightarrow^* Q'$ and $\chi \in \mathit{SBarbs}(Q')$. By the second condition of barbed bisimulation, there exists P' such that $P \longrightarrow^* P'$ and $Q' \preceq \mathbf{Er}_\Gamma(P')$. By Lemma D.14, $\chi \in \mathit{Barbs}(P')$. So, we have $\chi \in \mathit{Barbs}(P)$ as required. □

□