Secure Information Flow as Typed Process Behaviour*

Kohei Honda[1] Vasco Vasconcelos[2]
Nobuko Yoshida[3]

[1] Queen Mary and Westfield College, London, U.K.
[2] University of Lisbon, Lisbon, Portugal.
[3] University of Leicester, Leicester, U.K.

Abstract. We propose a new type discipline for the π-calculus in which secure information flow is guaranteed by static type checking. Secrecy levels are assigned to channels and are controlled by subtyping. A behavioural notion of types capturing causality of actions plays an essential role for ensuring safe information flow in diverse interactive behaviours, making the calculus powerful enough to embed known calculi for type-based security. The paper introduces the core part of the calculus, presents its basic syntactic properties, and illustrates its use as a tool for programming language analysis by a sound embedding of a secure multi-threaded imperative calculus of Volpano and Smith. The embedding leads to a practically meaningful extension of their original type discipline. Other fundamental technical elements, culminating in the behavioural non-interference result, are also sketched.

1. Introduction

In present-day computing environments, a user often employs programs which are sent or fetched from different sites to achieve her/his goals, either privately or in an organisation. Such programs may be run as a code to do a simple calculation task or as interactive parallel programs doing IO operations or communications, and sometimes deal with secret information, such as private data of the user or classified data of the organisation. Similar situations may occur in any computing environments where multiple users share common computing resources. One of the basic concerns in such a context is to ensure programs do not leak sensitive data to the third party, either maliciously or inadvertently. This is one of the key aspects of the security concerns, which is often called *secrecy*. Since it is difficult to dynamically check secrecy at run-time, it may as well be verified statically, i.e. from a program text alone [12]. The *information flow analysis* [12, 17, 37] addresses this concern by clarifying conditions when flow of information in a program is safe (i.e. high-level information never flows into low-level channels). Recent studies [2, 42, 50, 51, 53] have shown how we can integrate the techniques of type inference in programming languages with the ideas of information flow analysis, accumulating the basic principles of compositional static verification for secure information flow.

The study of type-based secrecy so far has been done in the context of functional or imperative calculi that incorporate secrecy. Considering that concurrency and communication are a norm in modern programming environments, one may wonder whether a similar study is possible in the framework of process calculi. There are two technical reasons why such an endeavour can be interesting. First, process calculi have been accumulating mathematically rigorous techniques to reason about computation based on communicating processes. In particular, given that an equivalence on program phrases plays a basic role for semantic justification of a type discipline for secrecy, cf.[53, 45], the theories of behavioural equivalences, cf. [21, 23, 38], which are a cornerstone in the study of process calculi, would offer a semantic

---

[*] The extended abstract to appear in Proc. of European Symposium on Programming (ESOP) 2000, LNCS, Springer, 2000.

basis for safe information flow in communicating processes. Second, type disciplines for communicating processes are widely studied recently, especially in the context of name passing process calculi such as the π-calculus, e.g. [11, 25, 32, 43, 49, 48, 55]. Further, recent studies have shown that name passing calculi enjoy great descriptive power, uniformly representing diverse language constructs as name passing processes, including those of imperative, functional and object-oriented languages, as well as sequential and concurrent, cf. [39, 54]. These representations are in close relationship with those given in games semantics, cf. [31, 30, 13], which suggest precise type structures in which the corresponding representation in name passing processes can be articulated. Since many real-life programming languages are equipped with diverse constructs from different paradigms, it would be interesting to see whether we can obtain a typed calculus based on name passing in which information flow involving various language constructs can be analysable on a uniform syntactic basis.

Against these backgrounds, the present work introduces a typed π-calculus in which secure information flow is guaranteed by static typing. Secrecy levels are attached to channels, and a simple subtyping ensures that interaction is always secrecy-safe. Information flow in this context arises as transformation of interactive behaviour to another interactive behaviour. Thus the essence of secure information flow becomes that a low-level interaction never depends on a high-level (or incompatible-level) interaction. Interestingly, this interaction-based principle of secure information flow strongly depends on the given type structures as *prerequisites*: that is, even semantically, certain behaviours can become either secure or insecure according to the given types. This is because types restrict a possible set of behaviours (which *are* information in the present context), thus affecting the notion of safe information flow itself. For this reason, a strong type discipline for name passing processes for linear and deadlock-free interaction [11, 32, 55] plays a fundamental role in the present typed calculus, by which we can capture safety of information flow in a wide range of computational behaviours, including those of diverse language constructs. This expressiveness can be used to embed and analyse typed programming languages for secure information flow. In this paper we explore the use of the calculus in this direction through a sound embedding of a secure multi-threaded imperative calculus of Volpano and Smith [50]. The embedding offers an analysis of the original system in which the underlying observable scenario is made explicit and is elucidated by typed process representation. As a result, we obtain a practically meaningful extension of the original type discipline with enlarged typability. We believe this example suggests a general use of the proposed framework, given the fundamental importance of the notion of observables in the analysis of secure computing systems [37, 50, 52].

Related work. Technically speaking, our work follows, on the one hand, Abadi's work on type-based secrecy in the π-calculus [1] (which is in turn based on [4]) and the studies on secure information flow in CCS and CSP (cf. [14, 36, 47]), and, on the other, the preceding works on type disciplines for name passing processes. In comparison with [1], the main novelty of the present typing system is that it ensures safety of information flow for general process behaviours rather than that for ground values, which is essential for the embedding of securely typed language constructs in the calculus. Compared to [14, 36, 47], a key difference lies in the fundamental role type information plays in the present system for defining and guaranteeing secrecy. Further, these works are not aimed at ensuring secrecy via static typing. Detailed comparison of the notion of secure information flow in the present work with those in [14, 36, 47], is an important further topic. Other notable works on the study of security-related aspects of name passing processes in general include [4, 9, 10, 22]. Among them, Bodei, Degano, Nielson and Nielson [9] utilise a control flow analysis for ensuring certain secrecy preservation. Hennessy and Reily [22] develop a typed π-calculus addressing integrity concerns where, as in the present work, each channel is assigned a security level. Our work differs from theirs in that we use the assignment of channels for secrecy concerns. It is an important subject for future study how we can seamlessly integrate these two concerns in a practically meaningful way.

In the context of type disciplines for name passing processes, the full use of dualised and directed types (cf. § 3), as well as their combination with causality-based dynamic types, is new, though the ideas are implicit in [16, 5, 24, 55, 15]. Our construction is based on graph-based types in [55], incorporating the partial algebra of types from [25] (the basic idea of modalities used here and in [25] originally come from linear logic [16]). The syntax of the present calculus is based on [48], among others branching and recursion. We use the synchronous version since it gives a much simpler typing system. The branching and recursion not only offer high-level abstraction useful for describing behaviour, but also are essential from the viewpoint of type discipline itself, as we shall discuss in Section 3. The calculus is soundly embeddable into the asynchronous π-calculus (also called the ν-calculus [28, 29]) by concise encoding [48]. The operational feasibility of branching and recursion is further studied in [35] (a variant of the calculus is also used in a recent study on types for distributed software [15]). For non-deterministic secrecy in general, security literature offers many studies based on probabilistic non-interference, cf. [19, 36, 51]. The incorporation of probability becomes necessary because if we can observe time being spent on program phrases then unsafe information flow can occur based on the probability distribution of non-deterministic states induced by scheduling. The present calculus and its theory are introduced as a basic stratum for the study of secure information flow in typed name passing processes, focussing on a simpler realm of possibilistic settings. Incorporation of the probability distribution in behavioural equivalences (cf. [34, 46]), as well as how this can lead to a useful framework for reasoning about secure behaviour of realistic programs with respect to observables with timing information, is an important subject of future study.

In a different vein, the dependency core calculus (DCC) by Abadi, Banerjee, Heintze and Riecke [2] presents a general calculus of functional dependency in which many secure (or causality-based) typed sequential calculi can be soundly embedded. The system is simple, yet it enjoys a great expressive power. In comparison, the present calculus is more complex while incorporating concurrency and communication as basic elements. The complexity arises essentially because it analyses a repertoire of computational behaviours which is broader than the pure (call-by-name) functional behaviour. We believe that, for analysing purely functional behaviours (or the behaviours which are easily embeddable into them), DCC would offer a clean and basic tool; while if non-pure elements such as non-determinism and exception are present, the presented calculus and its variants may often offer a more feasible framework for analysis. An important element of DCC is its clean denotational semantics: [45] also used powerdomains to formulate the semantic notion of secrecy. Our sequel offers a similar account for the present calculus in the form of bisimilarities. Our behavioural theories, which are briefly discussed in Section 5, are operational, which would be regarded as both strong and weak points. Behavioural theories often lack mathematical structure which their denotational counterparts own. However they are useful for reasoning about both sequential and concurrent computation: in particular, for concurrent computation, the behavioural theories in general have a merit of offering a wide range of equivalences suitable to a particular observational scenario. It is notable that some of the typing rules for the presented typed calculus itself were developed referring to the behavioural notion of safe information flow. This aspect will be developed in a later study.

As we already noted, one of the main results of the present work is an embedding of the multi-threaded imperative calculus by Volpano and Smith [50] in our typed π-calculus. The notion of secrecy level in this and other pioneering works by Volpano, Smith and Irvine, is closely related to that of the present calculus, in the sense that both represent the lowest level of tampering at which the program/process would affect the environment. Comparing their systems and ours at the level of typing, one significant character of the present framework is that it is based on a fine-grained notion of interaction where observables are represented explicitly. This can result in a new insight when we translate existing typed secure calculi into the present one, as we shall discuss in Section 6, taking Volpano-Smith's multi-threaded calculus. On the other hand, their framework is more useful than ours for directly reasoning about secrecy in imperative computation *per se.* Relatedly we note that their work treats other notions in imperative secrecy, such as polymorphic imperative procedure calls [53]. There are various imperative constructs such as continuations and exceptions. How the present work can be extended to these constructs is an important topic for further study.

For clear presentation of technical ideas, we divide the exposition of the technical elements into three parts. In **Part I**, which is this paper, we concentrate on the core part of the calculus and its basic syntactic properties, and illustrate its use for language analysis through the embedding of the multi-threaded calculus by Volpano and Smith. This gives a basic overview of the theory and its use. In **Part II** (to appear as [8]), we present the theory of secrecy-sensitive behavioural equivalences and discuss significant properties of typed terms, using the core calculus given in Part I. This leads to the establishment of: (1) the behavioural non-interference for the calculus, and (2) via (1), the non-interference results for the Volpano-Smith calculus. In **Part III**, we shall deal with a few extensions of the calculus, including free name passing and the corresponding behavioural theories.

Outline of this paper. In the remainder, Section 2 informally illustrates the basic ideas using examples. Section 3 introduces types and secrecy subtyping, as well as action types. Section 4 presents the type system and basic examples of typed processes. Section 5 discusses essential syntactic properties of the typed terms. Sections 6 gives the embedding result, and discusses how the result suggests an extension to the original typing system for the imperative calculus. Appendix A, B, C and D give auxiliary definitions, including the syntax-directed typing system. Appendix E gives the proofs omitted in the main sections. The present paper is the full version of the extended abstract [27], offering all proofs of the formally stated results.

Acknowledgement. We deeply thank anonymous referees of ESOP 2000 for their useful comments on the condensed version of this paper. Referring to their criticisms, many discussions/illustrations on technical and conceptual points are added in the present version. We deeply thank Gavin Lowe for his reading the early version of this paper. Our thanks also go to Edmund Robinson and Peter O'Hearn for their penetrating questions on the occasion of Tea Talk in November at QMW. Edmund also offered criticisms on the initial draft. We also had a nice discussion with Pasquale Malacaria. Our thanks go to Martin Berger for our collaboration on security-sensitive behavioural theories.

2. Basic Ideas

2.1. A Simple Principle

Let us consider how the notion of information flow arises in interacting processes, taking a simplest example. A CCS term $a.\overline{b}.\mathbf{0}$ represents a behaviour which synchronizes at a as input, then synchronizes at b as output, and does nothing. Suppose we attach secrecy levels to each port, for example "High" to a and "Low" to b. Intuitively this means that we wish interaction at a to be secret, while interaction at b may be known by a wider public: any high-level security process may interact at a and b, while a low-level security process can interact only at b. Then this process represents insecure interactions: any process observing b, which can be done by a low-level process, has the possibility to know an interaction at a, so information is indeed transmitted to a lower level from a higher level. Note that this does not depend on a being used for input and b used for output: $\overline{a}.b.\mathbf{0}$ with the same assignment of secrecy levels is similarly unsafe. In both cases, we are saying that if there is a causal dependency from an action at a high-level channel to the one at a low-level channel, the behaviour is not safe from the viewpoint of information flow (by causal dependency between two actions we mean, roughly speaking, that the dependent action cannot take place without the presence of the preceding action). Further, if we have value passing in addition, we would naturally take dependency in terms of communicated values into consideration.

The above informal principle based on causal dependency among interaction[1] is simple,

---

[1] Related ideas are studied in the context of CCS [14] and CSP [47].

but may look basic as a way of stipulating information flow for processes. Since many language constructs are known to be representable as interacting processes [6, 7, 30, 31, 38, 39], one may wonder whether the above idea can be used for understanding safety in information flow in various programming languages. In the following, we consider this question by taking basic examples of information flow in imperative programs.

2.2. Syntax

Let $a, b, c, \ldots x, y, z, \ldots$ range over *names* (which are both points of interaction and values to be communicated), and $X, Y, \ldots$ over *agent variables.* We write $\vec{y}$ for a vector of names $y_0 \cdots y_{n-1}$ with $n \geq 0$. Then the syntax for *processes*, written $P, Q, R, \ldots$, is given by the following grammar. We note that this syntax extends the standard polyadic π-calculus with branching and recursion.[2]

$$\begin{array}{rclllll}
P & ::= & x(\vec{y}).P & \text{input} & \mid & P \mid Q & \text{parallel}\\
& \mid & \overline{x}\langle(\boldsymbol{\nu}\,\vec{z})\vec{y}\rangle.P & \text{output} & \mid & (\boldsymbol{\nu}\,x)P & \text{hiding}\\
& \mid & x[(\vec{y}).P \,\&\, (\vec{z}).Q] & \text{branching input} & \mid & \mathbf{0} & \text{inaction}\\
& \mid & \overline{x}\,\mathtt{inl}\langle(\boldsymbol{\nu}\,\vec{z})\vec{y}\rangle.P & \text{left selection} & \mid & X\langle\vec{x}\rangle & \text{recursive variable}\\
& \mid & \overline{x}\,\mathtt{inr}\langle(\boldsymbol{\nu}\,\vec{z})\vec{y}\rangle.P & \text{right selection} & \mid & (\mu X(\vec{x}).P)\langle\vec{y}\rangle & \text{recursion}
\end{array}$$

Note there are two kinds of inputs, one unary and another binary: the former is the standard input in the π-calculus, while the latter, the *branching input*, has two branches, waiting for one of them to be selected with associated communication [48]. Accordingly there are outputs with left and right selections, as well as the standard one. We require all vectors of names in round parenthesis are pairwise distinct, which act as binders. In the value part of an output (including selections), say $\langle(\boldsymbol{\nu}\,\vec{z})\vec{y}\rangle$, names in $\vec{z}$ should be such that $\{\vec{z}\} \subset \{\vec{y}\}$ ($\{\vec{x}\}$ is the set of names in $\vec{x}$), and the order of occurrences of names in $\vec{z}$ should be the same as the corresponding names in $\vec{y}$. In these outputs, $(\boldsymbol{\nu}\,\vec{z})$ indicate names $\vec{z}$ are new names and are exported in the output actions. $\langle(\boldsymbol{\nu}\,\vec{z})\vec{y}\rangle$ is written $\langle\vec{y}\rangle$ if $\vec{z} = \emptyset$, and $(\boldsymbol{\nu}\,\vec{z})$ if $\vec{y} = \vec{z}$. We often omit vectors of the length zero (for example, we write $\mathtt{inr}$ for $\mathtt{inr}\langle\ \rangle$) and the trailing $\mathbf{0}$. The binding and α-convertibility $\equiv_\alpha$ are defined in the standard way. In a recursion $(\mu X(\vec{x}).P)\langle\vec{y}\rangle$, we require that P is *input guarded*, that is P is either a unary input or a branching input, and free names in P are a subset of $\{\vec{x}\}$. The reduction relation $\longrightarrow$ is defined in the standard manner, which we informally explain below by examples and whose formal definition is given in Appendix A.

We illustrate the syntax by examples. First, the following agents represent a boolean constant denoting the truth and the corresponding conditional selection, respectively (let c and y be fresh).

$$\mathbf{T}\langle b\rangle \quad = \quad b(c).(\overline{c}\,\mathtt{inl} \mid \mathbf{T}\langle b\rangle) \qquad \text{and} \qquad \mathbf{If}\langle x,\ P,\ Q\rangle \overset{\text{def}}{=} \overline{x}(\boldsymbol{\nu}\,y).y[().P \,\&\, ().Q]$$

The recursive definition of $\mathbf{T}\langle b\rangle$ is a notational convention and in fact stands for $\mathbf{T}\langle b\rangle \overset{\text{def}}{=} (\mu X(b).b(c).(\overline{c}\mathtt{inl} \mid X\langle b\rangle))\langle b\rangle$. The truth agent first inputs a name c via b, then, via c, does the left selection with no value passing as well as recreating the original agent. By replacing $\mathtt{inl}$ by $\mathtt{inr}$, we can define the falsity. The conditional process invokes a boolean agent, then waits with two branches. If the other party is truth it generates P: if else it generates Q. We can now show how these two processes interact:

$$\mathbf{If}\langle x,\ P,\ Q\rangle \mid \mathbf{T}\langle x\rangle \ \longrightarrow\ (\boldsymbol{\nu}\,y)(y\,[().P \,\&\, ().Q] \mid \overline{y}\,\mathtt{inl} \mid \mathbf{T}\langle x\rangle) \ \longrightarrow\ P \mid \mathbf{T}\langle x\rangle$$

Next we consider a representation of imperative variable as a process.

---

[2] These extensions are fundamental for the type discipline, in the sense that only in this way type structures and syntactic structures match: intended types are hard to deduce if we use the encoding of the high-level syntax into the polyadic π-calculus. Detailed discussions are given in Section 4 after we introduce the typing system. We also note we later restrict typed terms to those which only use bound name output, cf. §3.1. This section uses free name output since it is more convenient for illustrative purposes.

$$\mathbf{Var}\langle xv\rangle = x[(z).(\overline{z}\langle v\rangle \mid \mathbf{Var}\langle xv\rangle)\ \mathtt{\&}\ (v').\mathbf{Var}\langle xv'\rangle]$$

In this representation, we label the main interaction point of the process (called *principal port* in Interaction Net [33]) by the name of the variable (here x). It has two branches, of which the left one corresponds to the "read" option, while the right one corresponds to the "write" option. If the "read" is selected and z is received, the process sends the current value v to z, while regenerating the original self. On the other hand, if the "write" branch is selected and v' is received, then the process regenerates itself with a new value v'. We can then consider the representation of the assignment "$x := y$." This agent first "reads" the value from the variable y, then "writes" that value to the variable x.

$$\mathbf{Assign}\langle xy\rangle \overset{\mathrm{def}}{=} \overline{y}\,\mathtt{inl}(\boldsymbol{\nu}\,z).z(v).\overline{x}\,\mathtt{inr}\langle v\rangle$$

By letting the imperative variables and assignment interact, we have the following reduction sequence.

$$\begin{aligned}\mathbf{Var}\langle x1\rangle|\mathbf{Var}\langle y2\rangle|\mathbf{Assign}\langle xy\rangle \ &\longrightarrow\ \mathbf{Var}\langle x1\rangle|\mathbf{Var}\langle y2\rangle|(\boldsymbol{\nu}\,z)(\overline{z}(2)|z(v).\overline{x}\mathtt{inr}\langle v\rangle)\\ &\longrightarrow\ \mathbf{Var}\langle x1\rangle|\mathbf{Var}\langle y2\rangle|\overline{x}\mathtt{inr}\langle 2\rangle \ \longrightarrow\ \mathbf{Var}\langle x2\rangle|\mathbf{Var}\langle y2\rangle.\end{aligned}$$

2.3. Imperative Information Flow in Process Representation

(1) Causal Dependency. We can now turn to the information flow. We first consider the process representation of the following code, which gives an example of obviously insecure information flow [37].

$$x^{\mathrm{L}} := y^{\mathrm{H}}$$

Here the superscripts "L" and "H" indicate the secrecy levels of variables: thus y is a high (or secret) variable and x is a low (or public) variable. This command is insecure intuitively because the content of a secret variable becomes visible to the public through x. Following the previous discussion, its process representation becomes:

$$\mathbf{Assign}\langle x^{\mathrm{L}}y^{\mathrm{H}}\rangle \overset{\mathrm{def}}{=} \overline{y}^{\mathrm{H}}\,\mathtt{inl}(\boldsymbol{\nu}\,c).c^{\mathrm{H}}(v).\,\overline{x}^{\mathrm{L}}\,\mathtt{inr}\langle v\rangle \qquad (*)$$

Note we are labeling *channels* by secrecy levels. We can easily see that this process violates the informal principle we stipulated in § 2.1 because its low-level behaviour (at x) depends on its preceding high-level behaviour (at y, c). Thus this example does seem explainable from our general principle. Similarly, we can check the well-known example of implicit insecure information flow

$$\mathtt{if}\ z^{\mathrm{H}}\ \mathtt{then}\ x^{\mathrm{L}} := y^{\mathrm{L}}\ \mathtt{end}$$

which is insecure because the information stored in z can be indirectly revealed by reading x, is translated into insecure process interaction

$$\overline{z}^{\mathrm{H}}(\boldsymbol{\nu}\,c).c^{\mathrm{H}}[().\mathbf{Assign}\langle x^{\mathrm{L}}y^{\mathrm{L}}\rangle\ \mathtt{\&}\ ().\mathbf{0}] \qquad (**)$$

Note again the low-level interactions (in $\mathbf{Assign}\langle x^{\mathrm{L}}y^{\mathrm{L}}\rangle$) depend on the high-level interactions at z and c: thus insecurity in the original command is again understandable from the interaction-based viewpoint.

(2) Deadlock-Freedom. So far there has been no difficulty in applying our general principle to process presentation of imperative information flow. However there are subtleties to be understood, one of which already arises in the following simple sequential composition.

$$x^{\mathrm{H}} := y^{\mathrm{H}}\ ;\ z^{\mathrm{L}} := w^{\mathrm{L}}$$

The whole command is considered to be safe since whatever the content of x and y would be, they do not influence the content of z and w. However the following process representation of this command seems *not* safe in the light of our principle:

$$\overline{y}^{\mathrm{H}}\,\mathtt{inl}(\boldsymbol{\nu}\,c_1).c_1^{\mathrm{H}}(v_1).\,\overline{x}^{\mathrm{H}}\,\mathtt{inr}\langle v_1\rangle.\,\overline{w}^{\mathrm{L}}\,\mathtt{inl}(\boldsymbol{\nu}\,c_2).c_2^{\mathrm{L}}(v_2).\,\overline{z}^{\mathrm{L}}\,\mathtt{inr}\langle v_2\rangle \qquad (\star)$$

Here we find the situation where the behaviour at low-level ports (at w and z) depend on, via prefixing, the behaviour at high-level ports (at x and y). Does this mean that our principle and the standard idea in information flow are incompatible with each other? However, a closer look at the above representation reveals that this problematic dependency of low-level actions on high-level actions does not exist in effect, *provided* that the above process interacts with the processes for imperative variables which appeared in §2.2. If we assume so, then the actions at y and x (together with those at z and w) by the above process are always enabled: whenever a program wishes to access a variable, it always succeeds (in the interactional parlance, we are saying that interactions at these names are guaranteed to be *deadlock-free*). Thus we can guarantee that, under the assumption, the action at say w above will surely take place, which means the dependency as expressed in syntax does not exist indeed. Observing that there is no dependency at the level of communicated values between the two halves of ($\star$), we can now conclude that the actions at w and z do *not* causally depend on the preceding actions at y and x.

We note that, even if we loosen the notion of dependency by incorporating deadlock-freedom, the two translations which are found to be insecure in § 2.3 (1), ($*$) and ($**$), are still insecure, because, while the insecurity by dependency in synchronisation disappears, that by dependency in values persists.

(3) Innocuous Interaction. We now move to another subtle example, using the following command.

$$\mathtt{if}\ z^{\mathrm{H}}\ \mathtt{then}\ x^{\mathrm{H}} := y^{\mathrm{L}}\ \mathtt{end}$$

While this phrase is usually considered to be secrecy-wise safe [37], its process representation becomes:

$$\overline{z}^{\mathrm{H}}(\boldsymbol{\nu}\,c^{\mathrm{H}}).c[().\overline{y}^{\mathrm{L}}\mathtt{inl}(\boldsymbol{\nu}\,e).e^{\mathrm{L}}(v).\overline{x}^{\mathrm{H}}\,\mathtt{inr}\langle v\rangle \;\&\; ().\mathbf{0}] \qquad (\star\star)$$

which again shows apparently unsafe dependency between the second action at c and the third action at y (in the left branch). Note, in this example, the process does get information at c in the form of binary selection, even though c is deadlock-free. Moreover the output at y does not occur in the right branch, so that the output depends on the action at c even observationally. But the preceding study [53, 50] shows the original imperative behaviour is indeed safe, even in the multi-threaded setting. How can it be so? Simple, because this command only *reads* from y, without writing anything: so it is as if it did nothing to y. Returning to the process representation in ($\star\star$), we find the idea we made resort to in (2) above, is again effective: we consider this output action as not affecting the environment (hence not transmitting any information) *provided* that the behaviour of the interacting party in the environment is such that invoking its left branch has no real effect — in other words, if it behaves just as the imperative variable given in § 2.2 does. We call such an output *innocuous*: thus, if we decide to ignore the effect of innocuous actions, we find that there is no unsafe dependency from the high-level to the low-level (in fact the left branch as a whole now becomes high-level). We further observe that the insecure examples in (1) are still insecure even after the incorporation of the innocuousness. The safety principle discussed in § 2.1 looks still valid, explaining the information flow in the imperative behaviour from the interaction viewpoint.

The preceding discussions suggest two things: first, we may be able to formally stipulate the interactional framework of safe information flow which may have wide applicability along the line of the informal notion given in § 2.1. Secondly, however, just for that purpose, we need a non-trivial notion of types for behaviours which in particular concerns not only the behaviour of the process but also that of the assumed environment. The formal development in the following sections shows how these ideas can be materialised as a typed process calculus for safe information flow.

3. A Typed π-Calculus for Secure Information Flow (1) Types

3.1. Overview

In addition to *names* and *agent variables* (cf. §2.2), the typed calculus we introduce below uses a set of multiple *secrecy levels*, which are assumed to form a lattice.[3] $s, s', \ldots$ range over secrecy levels, and $s \leq s'$ etc. denotes the partial order (where the lesser means the lower, i.e. more public). Using these data as base sets, our objective in this section is to introduce a typing system whose provable sequent has the following form:

$\Gamma \vdash_s P \triangleright A$ "a process P has an action type A under a base Γ with a secrecy index s"

We offer an overview of the four elements in the above sequent.

(i) The *base* Γ is a finite function from names and agent variables to types and vectors of types, respectively. Intuitively if a type is assigned to a channel, then it denotes the basic structure of interaction at that channel, for example input or output, and selection or branching. For those with modes **!** and **?**, we also include refined modalities, which indicate whether they involve state change or not. When an agent variable is assigned a vector of types, this indicates a possible vector of names which can be used as a parameter to that agent variable.

(ii) The *process* P is an untyped term in § 2.2 which is annotated with types in its bound names, e.g. a unary input becomes $x(\vec{y}{:}\vec{\alpha}).P$ (here and elsewhere we assume $len(\vec{\alpha}) = len(\vec{y})$ where $len(\vec{y})$ denotes the length of a vector, so that each y_i is assigned a type α_i). As one notable aspect, we only use those processes whose outputs (in any of three forms) are *bound*, e.g. each unary output has a form $\overline{x}(\boldsymbol{\nu}\,\vec{y} : \vec{\alpha}).P$.[4] Accordingly we set names in each vector instantiating agent variables to be pairwise distinct. These restrictions make typing rules simpler, elucidate basic nature of the present notion of types, and give enough descriptive power to serve our present purpose. For reference, the syntax of processes is given in Appendix B. It should be noted here that branching and recursion are not only useful for high-level abstraction but play an essential role in the typing system, cf. Remark 4.1 later.

(iii) The *secrecy index* s in $\Gamma \vdash_s P \triangleright A$ guarantees that P under Γ only affects the process environment at levels at s or higher: that is, it is transmitting information only at levels no less than s. This "transmission of information" is best understood as affecting, or *tampering*, the environment. For this reason we often call a secrecy index a *tampering level*. Whether some action transmits information or not, is sometimes subtle, as we shall discuss as we introduce each typing rule below.

(iv) The *action type* A gives abstraction of the causal dependency among (actions on) free channels in P, ensuring, among others, certain deadlock-free properties on its linear and recursive channels. The activation ordering is represented by a partial order on nodes whose typical form is $\mathsf{p}x$ where p denotes a type of action to be done at x. There is a partial algebra over action types, by which we can control the composability of two action types (hence of typed processes which own them). This materialises the idea of

---

[3] As an intuition, consider the inverse lattice of subsets of a set $\{A, B, C\}$, where A, B, C are three users (so the top is $\emptyset$, the bottom is the whole set, and the order is the inverse of the subset inclusion). Then channels at level $\{A, B\}$ are those directly or indirectly readable by A and B; while $\top = \emptyset$ indicates a channel which is readable by nobody, or only by some (implicit) superuser. If an observer can interact at $\{A, B\}$ then s/he can surely interact at $\{A, B, C\}$, but not necessarily $\{B, C\}$: thus an observer who can interact at level s can always interact at level $s' \leq s$, but not necessarily at those levels compatible with s.

[4] This restricted output is an important mode of communication which arises both in the π-calculus [44] and in games semantics [7, 31, 30]. The essence of this mode lies in the potential to control the sharing of channels. This is fully realised with the present strong behavioural type discipline. This restricted mode of communication combined with the extended syntax substantially simplifies the typing rules. The introduced type structures are in close relationship with those arising in games semantics. The extension of type structures for free name passing, as well as its significance, will be discussed in Part III.

(Well-formedness and Compatibility)

$$\frac{-}{\vdash \tau} \quad \frac{\vdash \tau \asymp \tau'}{\vdash \langle \tau,\ \tau' \rangle} \quad \frac{\vdash \tau \asymp \tau'}{\vdash \tau' \asymp \tau} \quad \frac{\vdash \tau_i \asymp \tau'_i}{\vdash (\vec{\tau})^{\Downarrow}_{s} \asymp (\vec{\tau}')^{\Uparrow}_{s}} \quad \frac{\vdash \tau_i \asymp \tau'_i \quad s \geq s'}{\vdash (\vec{\tau})^{\downarrow}_{s} \asymp (\vec{\tau}')^{\uparrow}_{s'}} \quad \frac{\vdash \tau_i \asymp \tau'_i \quad s \geq s'}{\vdash (\vec{\tau})^{!}_{s,\kappa} \asymp (\vec{\tau}')^{?}_{s',\kappa}}$$

$$\frac{\vdash \tau_{ij} \asymp \tau'_{ij}}{\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\Downarrow}_{s} \asymp [\vec{\tau}'_1 \oplus \vec{\tau}'_2]^{\Uparrow}_{s}} \quad \frac{\vdash \tau_{ij} \asymp \tau'_{ij} \quad s \geq s'}{\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\downarrow}_{s} \asymp [\vec{\tau}'_1 \oplus \vec{\tau}'_2]^{\uparrow}_{s'}} \quad \frac{\vdash \tau_{ij} \asymp \tau'_{ij} \quad s \geq s'}{\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s,\kappa_1 \& \kappa_2} \asymp [\vec{\tau}'_1 \oplus \vec{\tau}'_2]^{?}_{s',\kappa_1 \oplus \kappa_2}}$$

(Subtyping)

$$\frac{\vdash \tau_i \leq \tau'_i}{\vdash (\vec{\tau})^{\Downarrow}_{s} \leq (\vec{\tau}')^{\Downarrow}_{s}} \quad \frac{\vdash \tau_i \leq \tau'_i \quad s \geq s'}{\vdash (\vec{\tau})^{\downarrow}_{s} \leq (\vec{\tau}')^{\downarrow}_{s'}} \quad \frac{\vdash \tau_i \leq \tau'_i \quad s \geq s'}{\vdash (\vec{\tau})^{!}_{s,\kappa} \leq (\vec{\tau}')^{!}_{s',\kappa}}$$

$$\frac{\vdash \tau_i \leq \tau'_i}{\vdash (\vec{\tau})^{\Uparrow}_{s} \leq (\vec{\tau}')^{\Uparrow}_{s}} \quad \frac{\vdash \tau_i \leq \tau'_i \quad s \leq s'}{\vdash (\vec{\tau})^{\uparrow}_{s} \leq (\vec{\tau}')^{\uparrow}_{s'}} \quad \frac{\vdash \tau_i \leq \tau'_i \quad s \leq s'}{\vdash (\vec{\tau})^{?}_{s,\kappa} \leq (\vec{\tau}')^{?}_{s',\kappa}}$$

$$\frac{\vdash \tau_{ij} \leq \tau'_{ij}}{\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\Downarrow}_{s} \leq [\vec{\tau}'_1 \& \vec{\tau}'_2]^{\Downarrow}_{s}} \quad \frac{\vdash \tau_{ij} \leq \tau'_{ij} \quad s \geq s'}{\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\downarrow}_{s} \leq [\vec{\tau}'_1 \& \vec{\tau}'_2]^{\downarrow}_{s'}} \quad \frac{\vdash \tau_{ij} \leq \tau'_{ij} \quad s \geq s'}{\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s,\kappa_1 \& \kappa_2} \leq [\vec{\tau}'_1 \& \vec{\tau}'_2]^{!}_{s',\kappa_1 \& \kappa_2}}$$

$$\frac{\vdash \tau_{ij} \leq \tau'_{ij}}{\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\Uparrow}_{s} \leq [\vec{\tau}'_1 \oplus \vec{\tau}'_2]^{\Uparrow}_{s}} \quad \frac{\vdash \tau_{ij} \leq \tau'_{ij} \quad s \leq s'}{\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\uparrow}_{s} \leq [\vec{\tau}'_1 \oplus \vec{\tau}'_2]^{\uparrow}_{s'}} \quad \frac{\vdash \tau_{ij} \leq \tau'_{ij} \quad s \leq s'}{\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{?}_{s,\kappa_1 \oplus \kappa_2} \leq [\vec{\tau}'_1 \oplus \vec{\tau}'_2]^{?}_{s',\kappa_1 \oplus \kappa_2}}$$

$$\frac{\vdash \langle \tau_1,\ \tau_2 \rangle \quad \vdash \tau \leq \tau_1 \text{ or } \vdash \tau \leq \tau_2}{\vdash \tau \leq \langle \tau_1,\ \tau_2 \rangle} \quad \frac{\vdash \langle \tau'_1,\ \tau'_2 \rangle \quad \vdash \tau_i \leq \tau'_i}{\vdash \langle \tau_1,\ \tau_2 \rangle \leq \langle \tau'_1,\ \tau'_2 \rangle}$$

Figure 1. Subtyping

composable environments which we discussed in Section 2. After the introduction of typing rules, Remark 4.1 gives further illustration of this point.

In the following, we introduce the necessary machinery one by one which is used in the typing system.

3.2. Types and Subtyping

We start with the set of *action modes*, denoted $m, m', ...$, whose underlying operational ideas are illustrated by the following table.

$\Downarrow$	Non-linear (non-deterministic) input	$\Uparrow$	Non-linear (non-deterministic) output
$\downarrow$	Truly linear input (truly once)	$\uparrow$	Truly linear output (truly once)
!	Recursive input (always available)	**?**	Zero or more output (always enabled)

The notations **!** and **?** come from Linear Logic [16], which first introduced these modalities. We also let $\kappa, \kappa', \ldots$, called *mutability indices*, range over $\{\boldsymbol{\iota}, \boldsymbol{\mu}\}$. Mutability indices indicate whether a recursive behaviour is stateful or not: for input, $\boldsymbol{\iota}$ denotes the lack of state, which we call *innocence*, cf. [31], while $\boldsymbol{\mu}$ means it may be stateful, that is it may change behaviour after the action; for output, $\boldsymbol{\iota}$ denotes innocuousness, that is the inputting party is innocent, while $\boldsymbol{\mu}$ denotes possible lack of innocuousness. Given these base sets, the grammar of *types*, denoted $\alpha, \beta, \ldots$, are given by:

$$
\begin{array}{lcl}
\alpha & ::= & \tau \quad | \ \langle \tau,\ \tau' \rangle \\
\tau & ::= & \alpha_{\mathtt{I}} \ | \ \alpha_{\mathtt{O}} \\
\alpha_{\mathtt{I}} & ::= & (\vec{\tau})^{\Downarrow}_{s} \ | \ (\vec{\tau})^{\downarrow}_{s} \ | \ (\vec{\tau})^{!}_{s,\kappa} \ | \ [\vec{\tau}_1 \& \vec{\tau}_2]^{\Downarrow}_{s} \ | \ [\vec{\tau}_1 \& \vec{\tau}_2]^{\downarrow}_{s} \ | \ [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s,\kappa_1 \& \kappa_2} \\
\alpha_{\mathtt{O}} & ::= & (\vec{\tau})^{\Uparrow}_{s} \ | \ (\vec{\tau})^{\uparrow}_{s} \ | \ (\vec{\tau})^{?}_{s,\kappa} \ | \ [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\Uparrow}_{s} \ | \ [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\uparrow}_{s} \ | \ [\vec{\tau}_1 \oplus \vec{\tau}_2]^{?}_{s,\kappa_1 \oplus \kappa_2}
\end{array}
$$

Types of form $\langle \tau,\ \tau' \rangle$ are *pair types*, indicating structures of interaction for both input and output, while others are *single types*, which are only for either input or output. We write $\mathsf{md}(\alpha)$ for the set of action modes of the outermost type(s) in α, e.g. $\mathsf{md}((\vec{\tau})^{m}_{s}) = \{m\}$ and $\mathsf{md}(\langle (\vec{\tau}_1)^{m_1}_{s_1}, (\vec{\tau}_2)^{m_2}_{s_2} \rangle) = \{m_1, m_2\}$. We often write $\mathsf{md}(\alpha) = m$ for $\mathsf{md}(\alpha) = \{m\}$. Similarly, we write $\mathsf{sec}(\tau)$ for the security level of the outermost type in τ, e.g. $\mathsf{sec}((\vec{\tau})^{m}_{s}) = s$. We define the *dual* of m, written $\overline{m}$, as: $\overline{\Downarrow} = \Uparrow$, $\overline{\Uparrow} = \Downarrow$, $\overline{\uparrow} = \downarrow$, $\overline{\downarrow} = \uparrow$, $\overline{\mathbf{!}} = \mathbf{?}$ and $\overline{\mathbf{?}} = \mathbf{!}$. Then the dual of a type α, denoted by $\overline{\alpha}$, is given by inductively dualising each action mode in α, as well as exchanging & and $\otimes$.

Among types, those with body $(\vec{\tau})$ correspond to unary input/output, those with body $[\vec{\tau}_1 \& \vec{\tau}_2]$ correspond to branching input (with or without recursion), and those with body $[\vec{\tau}_1 \oplus \vec{\tau}_2]$ correspond to output with selections. In say $(\tau)^{\downarrow}$ (neglecting a secrecy level), τ indicates the type of an action to be done at the newly imported channel by the input. See Remark 3.2 below for further illustration of nested types.

We say α is *well-formed*, written $\vdash \alpha$, if it is derivable from the rules in Figure 1, where we also define the *compatibility relation* $\asymp$ over single types. Note that all single types are well-formed. A pair type is well-formed iff its constituting single types are compatible. In compatibility, the secrecy level of the input is always the same as or higher than that of the output: in the case of non-linear types, however, they should be the same. See Remark 3.2 below for the illustration of the underlying ideas.

Next, let us say α is a *subtype of* β, denoted $\vdash \alpha \leq \beta$, if this sequent is derivable by the rules in Figure 1. The following properties of the subtyping relation are notable. The proof is given in Appendix E.1.1.

Lemma 3.1 (subtype ordering)

(1) *$\leq$ is a partial order.*

(2) *Let $\vdash \alpha \leq \alpha'$. Then* (a) $\mathsf{md}(\alpha) = \{\uparrow, \downarrow\}$ *implies* $\mathsf{md}(\alpha') = \{\uparrow, \downarrow\}$, (b) $\mathbf{?} \in \mathsf{md}(\alpha)$ *implies* $\mathbf{?} \in \mathsf{md}(\alpha')$, (c) $\mathbf{!} \in \mathsf{md}(\alpha)$ *implies* $\mathbf{!} \in \mathsf{md}(\alpha')$, *and* (d) *if α is nonlinear, then α' is nonlinear.*

(3) $\vdash \alpha_i \leq \beta_i$ $(i = 1, 2)$ *and* $\vdash \beta_1 \asymp \beta_2$ *imply* $\vdash \alpha_1 \asymp \alpha_2$.

(4) $\vdash \alpha$ *and* $\vdash \alpha' \leq \alpha$ *imply* $\vdash \alpha'$.

Some comments on types, subtyping and compatibility follow.

Remark 3.2 (nested types) Nested types denote what the process would do after exporting or importing new channels (hence covariance of subtyping on nested types): as an example, neglecting the secrecy and mutability, $x : ((\,)^{\downarrow})^{\uparrow}$ denotes the behaviour of doing a truly linear output at x exporting one single new name, and at that name doing a truly linear input without importing any name.

(secrecy levels, compatibility and subtyping) Since safe information flow should never go from a higher level to a lower level, a rule of thumb is that two types are compatible if such a flow is impossible. Thus, because a flow can occur in both ways at non-deterministic channels (cf. § 2.1: note, if x is a non-deterministic channel, the existence of an output, or an input, at x itself is information, since it is possible there is no such action at all), two non-linear types can be related only when they have the same secrecy level. On the other hand, for compatibility of linear types, we require that the inputting side is higher than the outputting side in secrecy levels, since the flow never comes from the inputting party (further, in truly linear unary types, even the outputting party does not induce flow, cf. footnote 5, page 15). This is because (1) the inputting side is always available, so interactibility-wise it gives no information, and (2) the information in terms of branching always comes from the outputting side. Accordingly, the subtyping is covariant for output and contravariant for input with respect to secrecy levels.

(mutability index) As we explained already, the index $\boldsymbol{\iota}$ represents the recursive input behaviour without state change (innocence) or, dually, the output which does not tamper the corresponding recursive processes (innocuousness). $\boldsymbol{\mu}$ indicates the lack of innocence/innocuousness. Note such an index is only meaningful for recursive behaviours and their dual output. Naturally we stipulate that an innocent input can only be compatible with an innocuous output; and an innocent input can only be a subtype of an innocent input, and an innocuous output can only be a subtype of an innocuous output. While innocuousness and innocence do influence safety in information flow (in fact, an innocuous output does not tamper the environment at all, so that it is vacuous in terms of its secrecy level), we do not stipulate this at the level of subtyping, but reflect its property at the level of typing rules, which is more flexible, cf. §4.3 later.

3.3. Action Types

An *action type*, written $A, A', \ldots$, is a finite poset whose elements, called *action nodes*, are given by the grammar:

$$\mathbf{n} \ ::= \ \downarrow x \ \mid \ \uparrow x \ \mid \ \updownarrow x \ \mid \ \mathbf{!}x \ \mid \ \mathbf{?}x \ \mid \ \mathbf{?}^{\iota}x \ \mid \ \Updownarrow x \ \mid \ X\langle \vec{x} \rangle.$$

Among these expressions, $\updownarrow x$ indicates x is already used exactly once for both input and output, so that no more connection is possible at x. $\mathbf{?}^{\iota}x$ indicates that all actions occurring at x so far have been innocuous. $X\langle \vec{x} \rangle$ (with $len(\vec{x}) \geq 1$ always) indicates the point to which the behaviour recurs. $\Updownarrow$ indicates possibility of nondeterministic input and output. Other symbols have already been explained in the table in § 3.2.

Some notational conventions: we write $|A|$ for the set of action nodes in A, and $\leq_A$ (or simply $\leq$) for the partial order. We often write $\mathbf{n} \in A$ for $\mathbf{n} \in |A|$. $\mathsf{fn}(\mathbf{n})$ (resp. $\mathsf{fv}(\mathbf{n})$) denotes the set of names (resp. agent variables) in $\mathbf{n}$, and we define $\mathsf{sbj}(\mathbf{n})$ ("the subject of $\mathbf{n}$") by $\mathsf{sbj}(\mathsf{p}x) = \mathsf{sbj}(X\langle x\vec{y} \rangle) = x$. We also set $\mathsf{md}(\mathsf{p}x) \stackrel{\mathrm{def}}{=} \mathsf{p}$, while $\mathsf{md}(X\langle \vec{x} \rangle)$ is undefined.

An action type is often regarded as a directed graph whose nodes are $|A|$ and whose edges are s.t. there is an edge from $\mathbf{n}$ to $\mathbf{n}'$ (denoted $\mathbf{n} \to \mathbf{n}'$) iff $\mathbf{n} \lneq \mathbf{n}'$ and, moreover, for no $\mathbf{n}''$ we have $\mathbf{n} \lneq \mathbf{n}'' \lneq \mathbf{n}'$. For example, $\downarrow x \to \uparrow y$ says that a truly linear output at y becomes active just after a truly linear input at x. We only use those action types which conform to a well-formedness condition, which we stipulate below. In the typing rules, we use the following notations (let $\{x_i\}$ be free names in A).

$\downarrow\!\uparrow A$	A only contains $\downarrow x_i$ or $\uparrow x_i$	$A^{\text{-}x}$	x does not occur as subjects in A
$\mathbf{?}A$	A only contains $\mathbf{?}x_i$, $\mathbf{?}^{\iota}x_i$ or $\Updownarrow x_i$	$A \otimes B$	disjoint union, with $A \cap B = \emptyset$
$\mathbf{?}^{\iota}A$	A only contains $\mathbf{?}^{\iota}x_i$	$\overrightarrow{\mathsf{p}x}$	$\mathsf{p}_0x_0 \otimes \mathsf{p}_1x_1 \otimes \cdots \mathsf{p}_{n-1}x_{n-1}$ $(n \geq 0)$

We also say x is *active in A* if $\mathsf{p}x$ (for some p) is minimal in A.

Now write $\mathsf{fn}(A)$, $\mathsf{fv}(A)$ and $\mathsf{sbj}(A)$ for the sets of names, agent variables and subjects, respectively, of all nodes in A. We now stipulate:

(1) (operation on modes) A relation $\asymp$ is given by the least symmetric relation including $\downarrow \asymp \uparrow$, $\Updownarrow \asymp \Updownarrow$, and $\mathbf{!} \asymp \mathsf{p}$ and $\mathsf{p} \asymp \mathsf{p}$ with $\mathsf{p} \in \{\mathbf{?}, \mathbf{?}^{\iota}\}$. We also set, $\mathsf{p} \odot \mathsf{q}$, defined when $\mathsf{p} \asymp \mathsf{q}$, as the commutative partial operator generated by: $\uparrow \odot \downarrow = \updownarrow$, $\mathbf{!} \odot \mathbf{?} = \mathbf{!} \odot \mathbf{?}^{\iota} = \mathbf{!}$, $\mathbf{?}^{\iota} \odot \mathbf{?}^{\iota} = \mathbf{?}^{\iota}$, $\mathbf{?} \odot \mathbf{?} = \mathbf{?} \odot \mathbf{?}^{\iota} = \mathbf{?}$, and $\Updownarrow \odot \Updownarrow = \Updownarrow$.

(2) (well-formedness) A is *well-formed* when: (1) $\mathsf{sbj}(\mathbf{n}_1) \neq \mathsf{sbj}(\mathbf{n}_2)$ and $\mathsf{fv}(\mathbf{n}_1) \cap \mathsf{fv}(\mathbf{n}_2) = \emptyset$ if $\mathbf{n}_1 \neq \mathbf{n}_2 \in A$, and (2) $\mathbf{n}_i = \mathsf{p}_i x_i$ with $\mathsf{p}_i \in \{\downarrow, \uparrow\}$ $(i = 1, 2)$, if $\mathbf{n}_1 \lneq_A \mathbf{n}_2$.

(3) (coherence) Let A_1 and A_2 be well-formed. Then A_1 and A_2 are *coherent*, written $A_1 \asymp A_2$, when: (1) $\mathsf{fv}(A_1) \cap \mathsf{fv}(A_2) = \emptyset$; (2) if $x \in \mathsf{sbj}(A_1) \cap \mathsf{sbj}(A_2)$, then $\mathsf{p}x \in A_1$ and $\mathsf{q}x \in A_2$ such that $\mathsf{p} \asymp \mathsf{q}$; and (3) if $x, y \in \mathsf{sbj}(A_1) \cap \mathsf{sbj}(A_2)$, then $\mathsf{p}x \lneq_{A_i} \mathsf{q}y$ implies $\mathsf{q}'y \not\lneq_{A_j} \mathsf{p}'x$ for $i \neq j$.

Note that in the coherence, we prohibit the possibility of having a circular dependency among combined nodes by the condition (3), as well as stipulating all interacting nodes have compatible modes. Since composition of two action types is defined only when they are mutually coherent, the operator $\odot$ (together with other two operators) is *partial* [25], i.e. it is not everywhere defined. This partiality controls the composability of two typed processes in the typing system, representing the notion of composable environments (cf. §2.1). We can now introduce the operators on well-formed action types.

(i) (prefix) Assume $\mathsf{fn}(\mathbf{n}) \cap \mathsf{fn}(A) = \emptyset$ and that each node in A as well as $\mathbf{n}$ has a mode from $\{\uparrow, \downarrow\}$. Then $\mathbf{n} \to A$ is an action type with the new minimum node $\mathbf{n}$, otherwise keeping the original order.

(ii) (composition) Assume $A_1 \asymp A_2$. Then $A_1 \odot A_2$ is a action type defined by:

- (node) $|A_1 \odot A_2| \stackrel{\text{def}}{=} \{\ (\mathsf{p} \odot \mathsf{q})x \mid \mathsf{p}x \in A_1,\ \mathsf{q}x \in A_2\} \cup \{\ \mathbf{n} \mid \mathbf{n} \in A_i,\ \mathsf{sbj}(\mathbf{n}) \notin \mathsf{sbj}(A_j), i \neq j\}$.
- (order) $\leq_{A_1 \odot A_2}$ is generated by: (1) $(\mathbf{n} \leq_{A_1} \mathbf{n}'$ or $\mathbf{n} \leq_{A_2} \mathbf{n}') \ \Rightarrow\ \mathbf{n} \leq_{A_1 \odot A_2} \mathbf{n}'$, and (2) $(\mathbf{n} \leq_{A_i} \mathsf{p}x,\ \mathsf{q}x \leq_{A_j} \mathbf{n}', i \neq j) \ \Rightarrow\ \mathbf{n} \leq_{A_1 \odot A_2} \mathbf{n}'$, in both rules assuming $\mathbf{n}, \mathbf{n}' \in |A_1 \odot A_2|$.

(iii) (hiding) Let A be well-formed and $\vec{x}$ be *active in* A, i.e. each x_i of $\vec{x}$ is active in A. Then $A/\vec{x}$ denotes the result of taking off all nodes with subjects $\vec{x}$ from A, keeping the partial order on other nodes. $A/\vec{x}$ is undefined if $\vec{x}$ is not active in A.

Note that, in (ii), if $\mathsf{sbj}(A_1) \cap \mathsf{sbj}(A_2) = \emptyset$, then $A_1 \odot A_2$ coincides with the disjoint union $A_1 \otimes A_2$. We give some examples of action types and their composition.

Example 3.3

(i) Let $A \stackrel{\text{def}}{=} \uparrow x$, $B \stackrel{\text{def}}{=} \downarrow x \to \uparrow y \to \uparrow z$ and $C \stackrel{\text{def}}{=} \downarrow w \to \downarrow y$. Then: (1) $A \asymp B$, $B \asymp C$, $A \asymp C$; (2) $A \odot B = (\uparrow y \to \uparrow z) \otimes \updownarrow x$, $(A \odot B) \odot C = (\uparrow w \to \uparrow z) \otimes \updownarrow x \otimes \updownarrow y$; and (3) $B \odot C = (\{\downarrow x, \uparrow w\} \to \uparrow z) \otimes \updownarrow y$ (denoting the obvious graph), $A \odot (B \odot C) = (A \odot B) \odot C$.

(ii) Let $A \stackrel{\text{def}}{=} \mathbf{!}x \otimes \mathbf{?}y$ and $B \stackrel{\text{def}}{=} \mathbf{?}x \otimes \mathbf{?}y \otimes X\langle x \rangle$. Then $A \asymp B$, and $A \odot B = \mathbf{!}x \otimes \mathbf{?}y \otimes X\langle x \rangle$, while both $A \not\asymp A$ and $B \not\asymp B$.

(iii) If $A \stackrel{\text{def}}{=} \uparrow x \to \downarrow y$ then $A/x = \downarrow y$, while A/y is not defined.

Note that, in (i), even when we combine trees to generate a tree, the intermediate state can be a proper graph: this indicates that, if we wish to own a coherent algebra, we do need proper graphs even just for dealing with tree-like dependency structures.

The following lemma gives basic properties of operators. The proof is given in Appendix E.1.2.

Lemma 3.4 (well-definedness) *Assume A, A_1 and A_2 are well-formed.*

(1) *If $\mathbf{n} \to A$ is defined, then $\mathbf{n} \to A$ is well-formed.*

(2) *If $A/\vec{y}$ is defined, then $A/\vec{y}$ is well-formed.*

(3) *If $A_1 \asymp A_2$, then $A_1 \odot A_2$ is well-formed.*

The next lemma says $\odot$ is commutative and partially associative. See Appendix E.1.3 for the proof.

Lemma 3.5 (commutativity and associativity) *Let A_1, A_2, A_3 be well-formed.*

(i) *Assume $A_1 \asymp A_2$. Then we have $A_2 \asymp A_1$ and $A_1 \odot A_2 = A_2 \odot A_1$.*

(ii) *Assume $A_1 \asymp A_2$ and $(A_1 \odot A_2) \asymp A_3$. Then we have: (1) $A_1 \asymp A_3$ and $A_2 \asymp A_3$, (2) $A_1 \asymp (A_2 \odot A_3)$ and (3) $(A_1 \odot A_2) \odot A_3 = A_1 \odot (A_2 \odot A_3)$.*

(iii) *The empty graph $\emptyset$ is the unit for the operation $\odot$.*

Finally we note that the present class of action types only incorporate dependency among truly linear actions via the well-formedness condition. Thus the structure of action types is always in the form:

$$\underbrace{\overrightarrow{\Updownarrow x} \otimes \overrightarrow{\mathbf{?} z} \otimes \underbrace{\overrightarrow{\mathbf{?}^{\iota} v}}_{\mathbf{?}^{\iota} B}}_{\mathbf{?} C} \otimes \overrightarrow{\mathbf{!} w} \otimes \overrightarrow{\updownarrow y} (\otimes X\langle\vec{u}\rangle) \otimes \downarrow\uparrow E$$

where $(\otimes X\langle\vec{u}\rangle)$ shows this item may or may not exist. Extending this simple form, there are diverse possibilities to represent causality among actions, some of which are being explored by the present authors.

4. A Typed π-Calculus for Secure Information Flow (2) Typing System

4.1. Typing System (1) Process Composition and Weakening

We now introduce the main typing rules with illustration (the whole table appears in Appendix C). We use the following notation: (1) given a base Γ (cf. § 3.1), (1) $x : \alpha$ (resp. $X : \vec{\alpha}$) denotes $\Gamma(x) = \alpha$ (resp. $\Gamma(X) = \vec{\alpha}$); and (2) $\Gamma \cdot \Delta$ denotes the disjoint union of two bases, assuming their domains do not intersect. We also henceforth assume all types and bases are well-formed. We start from the typing rules for basic process operators: the inaction, parallel composition and hiding.

$$\text{(Zero)}\ \frac{}{\Gamma \vdash_s \mathbf{0} \triangleright \emptyset} \qquad \text{(Par)}\ \frac{A_1 \asymp A_2 \quad \Gamma \vdash_s P_i \triangleright A_i \quad (i = 1, 2)}{\Gamma \vdash_s P_1 \mid P_2 \triangleright A_1 \odot A_2} \qquad \text{(Res)}\ \frac{\Gamma \cdot x : \alpha \vdash_s P \triangleright A \otimes \mathrm{p}x \quad \mathrm{p} \in \{\updownarrow, \mathbf{!}, \Updownarrow\}}{\Gamma \vdash_s (\boldsymbol{\nu}\, x : \alpha) P \triangleright A}$$

Note, in (Par), we use the coherence $\asymp$ and the composition $\odot$, which we introduced in Section 3. In (Res), we do not allow a name with a mode in $\{\downarrow, \uparrow, \mathbf{?}, \mathbf{?}^{\iota}\}$ to be restricted since these actions expect their complementary actions to get composed — in other words, actions with these types assume the existence of actions with their dual types in the environment. With the complementary actions left uncomposed, the hiding leads to an insecure system.

We next present various weakening rules.

$$\text{(Deg}_s\text{)}\ \frac{\Gamma \vdash_s P \triangleright A \quad s' \leq s}{\Gamma \vdash_{s'} P \triangleright A} \qquad \text{(Weak-}\mathbf{?}^{\iota}\text{)}\ \frac{\Gamma \vdash_s P \triangleright A^{\text{-}x} \quad \mathbf{?} \in \mathsf{md}(\Gamma(x))}{\Gamma \vdash_s P \triangleright A \otimes \mathbf{?}^{\iota} x} \qquad \text{(Weak-}\mathbf{?}\text{)}\ \frac{\Gamma \vdash_s P \triangleright A \otimes \mathbf{?}^{\iota} x}{\Gamma \vdash_s P \triangleright A \otimes \mathbf{?} x}$$

$$\text{(Weak-}\Updownarrow\text{)}\ \frac{\Gamma \vdash_s P \triangleright A^{\text{-}x} \quad \Gamma(x) \text{ nonlinear}}{\Gamma \vdash_s P \triangleright A \otimes \Updownarrow x} \qquad \text{(Weak-}\updownarrow\text{)}\ \frac{\Gamma \vdash_s P \triangleright A^{\text{-}x} \quad \mathsf{md}(\Gamma(x)) = \{\downarrow, \uparrow\}}{\Gamma \vdash_s P \triangleright A \otimes \updownarrow x}$$

The degradation rule (Deg$_s$) should be natural considering our account in § 3.1 (iii): it says that if P under Γ is guaranteed to tamper only at the level s or higher, then P is also guaranteed to tamper only at the level $s' \leq s$ or higher. (Weak-$\mathbf{?}^{\iota}$) makes sense because $\mathbf{?}^{\iota} x$ indicates zero or more innocuous output actions at x: even when the type of x at Γ does not say it is innocuous, if no action has taken place yet on x (by $A^{\text{-}x}$), $\mathbf{?}^{\iota} x$ is meaningful (regarded as saying "zero innocuous actions"). On the other hand, it is perfectly fine to do the subsumption of $\mathbf{?}^{\iota} x$ by $\mathbf{?} x$, since the latter is more generous (i.e. it allows non-innocuous questions too). Note that, by this, the rule which adds $\mathbf{?}x$ in place of $\mathbf{?}^{\iota}x$ in (Weak-$\mathbf{?}^{\iota}$) is an admissible rule of the typing system. (Weak-$\Updownarrow$) weakens a nondeterministic action, which is understood in the same way as (Weak-$\mathbf{?}^{\iota}$). In (Weak-$\updownarrow$), the lack of actions at truly linear x is equated to the situation where x is used exactly once for both input and output. This rule is necessary, for example, for the subject reduction theorem later.

4.2. Typing System (2) Non-Linear Prefix Rules

The rules for prefix actually control the secrecy levels of each action. We start with non-linear prefixes.

$$\frac{\begin{array}{l}(\mathsf{In}) \quad \vdash (\vec{\tau})^{\Downarrow}_{s} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}\!:\!\vec{\tau} \vdash_{s} P \triangleright \overrightarrow{\mathsf{p}y} \otimes \mathbf{?}A \otimes \Updownarrow x\end{array}}{\Gamma \vdash_{s} x(\vec{y}\!:\!\vec{\tau}).P \triangleright A \otimes \Updownarrow x} \qquad \frac{\begin{array}{l}(\mathsf{Out}) \quad \vdash (\vec{\tau})^{\Uparrow}_{s} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}\!:\!\vec{\tau} \vdash_{s} P \triangleright \overrightarrow{\mathsf{p}y} \otimes \mathbf{?}A \otimes \Updownarrow x\end{array}}{\Gamma \vdash_{s} \overline{x}(\boldsymbol{\nu}\,\vec{y}\!:\!\vec{\alpha}).P \triangleright A \otimes \Updownarrow x}$$

Since the subtyping on non-linear types is trivial w.r.t. secrecy levels, $\vdash (\vec{\tau})^{\Uparrow,s} \leq \Gamma(x)$ means $\Gamma(x)$ has precisely the level s. Thus, in both (In) and (Out), the initial action at level s is followed by actions affecting the same or higher levels (because P is typed with s). Note also all abstracted action nodes ($\overrightarrow{\mathsf{p}y}$ above) should be active, which is essential for the subject reduction, cf. § 4. Non-linear prefix rules for branching and selections are essentially the same. For recursion, we have the following rule.

$$\frac{\begin{array}{l}(\mathsf{Var}) \quad \vdash \alpha_i \leq \Gamma(x_i) \qquad \mathsf{md}(\alpha_0) = \Downarrow \\ \begin{cases} \mathsf{p}_i = \Updownarrow & (\alpha_i \text{ nonlinear}) \\ \mathsf{p}_i = \mathsf{md}(\alpha_i) = \mathbf{?} & (\text{else}) \end{cases}\end{array}}{\Gamma \cdot X\!:\!\vec{\alpha} \vdash_{\mathsf{sec}(\alpha_0)} X\langle\vec{x}\rangle \triangleright \mathsf{p}\vec{x}} \qquad \frac{\begin{array}{l}(\mathsf{Rec}) \\ \vdash \alpha_i \leq \Gamma(z_i) \;\; \alpha_0 \text{ nonlinear} \\ \Gamma\{\vec{x}/\vec{z}\} \cdot X\!:\!\vec{\alpha} \vdash_{s} P \triangleright \mathbf{?}A\{\vec{x}/\vec{z}\}\end{array}}{\Gamma \vdash_{s} (\mu X(\vec{x}\!:\!\vec{\alpha}).P)\langle\vec{z}\rangle \triangleright A}$$

In (Var), the type α_0 corresponds to the subject of the initial prefix of the recursion (which is to bind X later in the proof tree). Note also, again by the subtyping rule, $\mathsf{sec}(\alpha_0)$ coincides with $\mathsf{sec}(\Gamma(x_0))$. Remembering the level should always elevate in a non-linear prefix, i.e. the body should be higher in secrecy levels than the subject of the prefix, if the tampering level of x_0 is not recorded, then we can have a term like $(\mu X(xy).x^{\mathrm{L}}.y^{\mathrm{H}}.X\langle xy\rangle)\langle xy\rangle$ (inferable if we stipulate $X\langle xy\rangle$ to be high), which obviously violates the safe information flow. This is why the level of x_0 should be recorded. Also note that, in the rule (Var), we place those names which are the parameter to the variable in the action type: this is because we need to take into account the existence of possible non-innocuous or non-deterministic actions which may occur when the variable is instantiated into the real behaviour.

In (Rec), we bind zero or more occurrences of X in P. By the rule (Var), if X ever occurs inside P, then the level must have stayed unchanged from the introduction of X by the (Var) rule to the binding of X by the (Rec) rule. We also note that, in this and other recursion rules, we effectively use the fact that names in the parameter vector of each recursive expression should be pairwise distinct: thus $\{\vec{x}/\vec{z}\}$ is well-defined and, moreover, always denotes injective renaming.

4.3. Typing System (3) Linear Prefix Rules

Among linear prefix rules, the following shows a stark contrast with the non-linear (In) and (Out) rules.

$$\frac{\begin{array}{l}(\mathsf{In}^{\downarrow}) \quad (\text{where } C/\vec{y} = \downarrow\!\uparrow B) \\ \vdash (\vec{\tau})^{\downarrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}\!:\!\vec{\tau} \vdash_{s} P \triangleright \mathbf{?}A \otimes C^{\text{-}x}\end{array}}{\Gamma \vdash_{s} x(\vec{y}\!:\!\vec{\tau}).P \triangleright A \otimes \downarrow x \rightarrow B} \qquad \frac{\begin{array}{l}(\mathsf{Out}^{\uparrow}) \quad (\text{where } C/\vec{y} = \downarrow\!\uparrow B) \\ \vdash (\vec{\tau})^{\uparrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}\!:\!\vec{\tau} \vdash_{s} P \triangleright \mathbf{?}A \otimes C^{\text{-}x}\end{array}}{\Gamma \vdash_{s} \overline{x}(\boldsymbol{\nu}\,\vec{y}\!:\!\vec{\tau}).P \triangleright A \otimes \uparrow x \rightarrow B}$$

The notation $C/\vec{y}$ denotes the result of taking off nodes with names among $\vec{y}$, as well as stipulating the condition that each y_i should be active in C. We observe that the "true linearity" in these and later rules is stronger than those studied in [25, 32], which only requires "less than once". In the rule, since s' is not given any condition in the antecedent, both rules completely neglect the secrecy level of x in Γ, saying we may not regard these actions as either receiving or giving information from/to the environment.[5] The operation $\mathbf{n}\!\rightarrow\! B$ records the causality.

The next rules show that branching/selection need a different treatment from the unary cases even if types are truly linear. Intuitively, the act of selection gives rise to a non-trivial flow of information.

$$\begin{array}{l}
(\mathsf{Bra}^{\downarrow}) \quad (\text{where } C_i/\vec{y}_i = {\downarrow}{\uparrow}B) \\
\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\downarrow}_{s} \leq \Gamma(x) \\
\Gamma \cdot \vec{y}_i : \vec{\tau}_i \vdash_s P_i \triangleright \mathbf{?}A \otimes C_i^{\text{-}x} \quad (i = 1, 2) \\
\hline
\Gamma \vdash_s x[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2] \triangleright A \otimes {\downarrow} x \rightarrow B
\end{array}
\qquad
\begin{array}{l}
(\mathsf{Sel}^{\uparrow}_{l}) \quad (\text{where } C/\vec{y}_1 = {\downarrow}{\uparrow}B) \\
\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\uparrow}_{s} \leq \Gamma(x) \\
\Gamma \cdot \vec{y}_1 : \vec{\tau}_1 \vdash_s P \triangleright \mathbf{?}A \otimes C^{\text{-}x} \\
\hline
\Gamma \vdash_s \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, \vec{y}_1 : \vec{\tau}_1).P \triangleright A \otimes {\uparrow} x \rightarrow B
\end{array}$$

Here the subtyping is used non-trivially: in (Bra$^{\downarrow}$), the real level of x in Γ is the same or lower than s, so the level elevates. In (Sel$^{\uparrow}$), the real level of x is the same or higher, so the level may go down, but it is recorded in the conclusion. It is notable that this inference crucially depends on the employment of branching as a syntactic construct: without it, these rules should have the same strict conditions as non-linear prefixes. See Remark 4.1 for further illustration of this point.

The final class of rules show the treatment of **!**-**?** modalities and mutability indices, dealing with recursive inputs and their dual outputs, and are most involved. We start with the variable introduction.

$$\begin{array}{l}
(\mathsf{Var}^{\mathbf{!}}) \qquad \vdash \alpha_i \leq \Gamma(x_i) \\
\mathsf{md}(\alpha_0) = \mathbf{!} \quad \mathsf{md}(\alpha_i) \in \{\mathbf{?}, \Downarrow, \Uparrow\} \ (i \neq 0) \\
\hline
\Gamma \cdot X : \vec{\alpha} \vdash_s X\langle \vec{x} \rangle \triangleright X\langle \vec{x} \rangle
\end{array}$$

The rule (Var$^{\mathbf{!}}$) places the variable together with its parameter in the action type. Note we give no restriction on s: when the introduced variable is later bound by a recursive prefix, all potential tampering at free names would have been recorded except the subject of this recursion (to be substituted for x_0). And this subject is never tampering, so does not count in terms of secrecy levels.

We can now move to the linear recursion rules. There are two pairs of rules, one for unary prefix and another for binary prefix. We start with the rules for unary input/output.

$$\begin{array}{l}
(\mathsf{In}^{\mathbf{!}}) \\
\vdash (\vec{\tau})^{\mathbf{!}}_{s,\kappa} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i) \\
\begin{cases}
\Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \vdash_s P \triangleright \overrightarrow{\mathrm{p}y} \otimes \mathbf{?}^{\boldsymbol{\iota}} A\{\vec{x}/\vec{z}\} \otimes X\langle \vec{x} \rangle & (\kappa = \boldsymbol{\iota}) \\
\Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \vdash_s P \triangleright \overrightarrow{\mathrm{p}y} \otimes \mathbf{?} A\{\vec{x}/\vec{z}\} \otimes X\langle x \vec{w} \rangle & (\kappa = \boldsymbol{\mu})
\end{cases} \\
\hline
\Gamma \vdash_s (\mu X(\vec{x} : \vec{\alpha}).x_0(\vec{y} : \vec{\tau}).P)\langle \vec{z} \rangle \triangleright \mathbf{!} z_0 \otimes A
\end{array}
\qquad
\begin{array}{l}
(\mathsf{Out}^{\mathbf{?}}) \quad (\text{where } C/\vec{y} = {\downarrow}{\uparrow}B) \\
\vdash (\vec{\tau})^{\mathbf{?}}_{s',\kappa} \leq \Gamma(x) \\
\Gamma \cdot \vec{y} : \vec{\tau} \vdash_s P \triangleright \mathbf{?}A \otimes C \otimes \mathrm{p}x \quad \mathrm{p} \in \{\mathbf{?}, \mathbf{?}^{\boldsymbol{\iota}}\} \\
\kappa = \boldsymbol{\mu} \Rightarrow (s = s' \wedge \mathrm{p} = \mathbf{?}) \\
\hline
\Gamma \vdash_s \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\tau}).P \triangleright A \otimes B \otimes \mathrm{p}x
\end{array}$$

In (In$^{\mathbf{!}}$), we check that the process is immediately recurring to precisely the same behaviour ($X\langle \vec{x} \rangle$) if it is innocent, or, if it is not innocent, it recurs to the same subject ($X\langle x_0 \vec{w}_j \rangle$). The process can only do free actions with $\mathbf{?}^{\boldsymbol{\iota}}$-modes in the innocent branch in addition to the recurrence (except at $\vec{y}$, which are immediately abstracted), so that the process is stateless

---

[5] These actions do transmit channels, but since they are bound they are only the *source* of information, not information itself. We can understand this by considering the representation of these actions in the standard early transition trees. In the representation, there is no branching up to alpha-conversion at actions at such channels (except those with interleaving actions), indicating, together with the fact that the action occurs exactly once in each trace, that information flow, or influence on different paths of behaviours, does not take place at that channel.

in its entire visible actions. In the conclusion, the new subject z_0 is introduced with the mode $\mathbf{!}$. In the dual $(\mathrm{Out}^{\mathbf{?}})$, if the prefix is an innocuous selection ($\kappa = \boldsymbol{\iota}$), there is no condition on the level of x (s'), so that the level is *not* counted either in the antecedent or in the conclusion (e.g. even if $s' = \bot$ we can have $s \neq \bot$): we are regarding the action as not affecting, and not being affected by, the environment (it is not affected since the inputting party is always available). However if the action is *not* innocuous ($\kappa = \boldsymbol{\mu}$), it is considered as affecting the environment (though the action is still unaffected by the environment), so that we record its secrecy level by requiring $s' = s$. Note that, even if it is unary, a $\mathbf{?}$-moded output action may indeed affect the environment simply because such an action may or may not exist: just as a unary non-deterministic input/output induces information flow.

We next turn to the version of the above two rules which involve branching/selection.

$$(\mathsf{Bra}^{\mathbf{!}})\qquad \frac{\begin{array}{l}\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\mathbf{!}}_{s,\kappa_1 \& \kappa_2} \le \Gamma(z_0) \quad \vdash \alpha_i \le \Gamma(z_i) \qquad (j=1,2)\\ \begin{cases} \Gamma\{\vec{x}/\vec{z}\}\cdot\vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \vdash_s P_j \triangleright \overrightarrow{\mathtt{p}y_j} \otimes \mathbf{?}^{\boldsymbol{\iota}} A_j\{\vec{x}/\vec{z}\} \otimes X\langle \vec{x}\rangle & (\kappa_j = \boldsymbol{\iota})\\ \Gamma\{\vec{x}/\vec{z}\}\cdot\vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \vdash_s P_j \triangleright \overrightarrow{\mathtt{p}y_j} \otimes \mathbf{?} A_j\{\vec{x}/\vec{z}\} \otimes X\langle x_0 \vec{w}_j\rangle & (\kappa_j = \boldsymbol{\mu})\end{cases}\end{array}}{\Gamma \vdash_s (\mu X(\vec{x}:\vec{\alpha}).x_0[(\vec{y}_1:\vec{\tau}_1).P_1 \;\&\; (\vec{y}_2:\vec{\tau}_2).P_2])\langle \vec{z}\rangle \triangleright \mathbf{!}z_0 \otimes (A_1 \odot A_2)}$$

$$(\mathsf{Sel}^{\mathbf{?}}_{l})\ \ (\text{where } C/\vec{y} = \downarrow\!\!\uparrow\ B)\qquad \frac{\begin{array}{l}\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\mathbf{?}}_{s',\kappa_1 \oplus \kappa_2} \le \Gamma(x)\\ \Gamma\cdot\vec{y}_1{:}\vec{\tau}_1 \vdash_s P \triangleright \mathbf{?}A \otimes C \otimes \mathtt{p}x \quad \mathtt{p} \in \{\mathbf{?}, \mathbf{?}^{\boldsymbol{\iota}}\}\\ \kappa_1 = \boldsymbol{\mu} \Rightarrow (s = s' \wedge \mathtt{p} = \mathbf{?})\end{array}}{\Gamma \vdash_s \overline{x}\mathtt{inl}(\boldsymbol{\nu}\,\vec{y}_1:\vec{\tau}_1).P \triangleright A \otimes B \otimes \mathtt{p}x}$$

In $(\mathrm{Bra}^{\mathbf{!}})$, which adds complexity, we again check whether the process is immediately recurring to the same subject ($X\langle x_0\vec{w}_j\rangle$) or, if moreover the branch is innocent, it recurs to precisely the same agent ($X\langle\vec{x}\rangle$). Note $\odot$ is used in the conclusion, which is necessary when there is one innocent branch and one non-innocent branch: in such cases, we cannot adjust the $\mathbf{?}^{\boldsymbol{\iota}}$-mode to the $\mathbf{?}$-mode in general since we do need $\mathbf{?}^{\boldsymbol{\iota}}$-mode for the innocent branch. Since if $\mathbf{?}A_1$ and $\mathbf{?}A_2$ are inferred under the same base Γ we always have $A_1 \asymp A_2$, the composition $A_1 \odot A_2$ is also well-defined. As an example of such composition, if $\mathbf{?}^{\boldsymbol{\iota}}x$ is in an innocent branch and $\mathbf{?}x$ is in a non-innocent branch, the result is $\mathbf{?}x \odot \mathbf{?}^{\boldsymbol{\iota}}x = \mathbf{?}x$, which is intuitively natural.

The rule $(\mathrm{Sel}^{\mathbf{?}})$ is the dual of $(\mathrm{Bra}^{\mathbf{!}})$ and can be understood as $(\mathrm{Out}^{\mathbf{?}})$. Note that, this time, information flow is induced both by the possible (non-)existence of the action and the selection of one of the branches.

Remark 4.1 (Representation of Environments) Section 2 discussed how the necessity to assume particular behaviours of the environment is important. The representation of environments in this sense is carried out in the present typing system by the partial algebra of action types via $\asymp$ and $\odot$. For example, if $\Gamma \vdash_s P \triangleright A \otimes \mathbf{!}x$, then we are assuming the environment has zero or more $\mathbf{?}$-actions at x but *not* $\mathbf{!}$-action at x. Thus $\Gamma \vdash_s Q \triangleright \mathbf{?}x$ is composable with the typed term, while $\Gamma \vdash_s Q' \triangleright \mathbf{!}x$ is not. This affects the notion of safety in information flow in the following way. A key criterion of the secrecy in the present context is "not to transmit the behaviour at high-level channels to low-level channels." If we restrict the channel x to be "output truly linear" for the environment, then the environment should *necessarily* output truly once at x: which means, if this is further unary, there is no information coming at x (since it is predetermined). What kind of information can come and go is determined by types, affecting the notion of safety in information flow.

(Nature of Typing Rules, 1) The present typing rules have a certain complexity in comparison with many foregoing typing systems for the π-calculus, as well as with the secrecy-based functional calculi such as DCC [2]. The complexity comes from two directions. First, the calculus deals with various kinds of behaviours, including deterministic and non-deterministic ones: typing rules should guarantee the secrecy for an arbitrary combination of non-deterministic, truly linear or recursively truly linear actions. Second, we are dealing with more general, or fine-grained, behaviours and their typed algebras than those of usual programming languages. For example, as we shall see in Section 6, an imperative secrecy calculus utilises a specific class of partial operators (e.g. sequencing) for guaranteeing secrecy. These are decomposed into name passing behaviours and their algebra in the present calculus. Thus typing rules treat arguably more fine-grained forms of typed algebras which can represent other forms of program composition.

(Nature of Typing Rules, 2) Another important aspect is that the present typing rules strongly depend on the syntactic structure we employed for the present calculus, in particular branching and recursion. A simple example is truly linear branching type. It is well-known that we can encode such branching structures into the polyadic (and indeed asynchronous monadic) π-calculus. However it is hard to obtain what corresponds to (Bra$^{\downarrow}$) by such an encoding. Take $x[().P_1 \,\&\, ().P_2]$, which is translated as $\overline{x}(\boldsymbol{\nu}\, c_1 c_2).(c_1.P_1 | c_2.P_2)$. The output selection becomes either $x(c_1 c_2).\overline{c_1}$ or $x(c_1 c_2).\overline{c_2}$. Note that, by the form of these agents, c_1 and c_2 should be non-linear: which immediately lowers the tampering level of the first agent to those of c_1 and c_2. The secrecy levels of c_1 and c_2 should be clearly lower than those in P_1 and P_2. Thus the outputting agents above have those tampering levels, which are in general lower than what we obtain in the present system. There is also difficulty in the control of linearity, which we do not discuss here. Similarly for the case of recursion, for which syntactic analysis of its encoding into replication using types would be harder even if we fix the form of the encoding. The appreciation of these points led to the choice and construction of the specific syntax in the present text.

(Variant Rules) We list a refinements of the above typing rules which result in added typability without changing the behavioural properties they guarantee, though they do make rules complex. Since innocuous actions never transmit information, dually, innocent actions never receive information. We can thus refine the above (In$^{!}$) rule thus:

$$\frac{\textsf{(In}^{!}\textsf{-variant)}\quad \vdash (\vec{\tau})^{!}_{s',\kappa} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i) \quad \begin{cases} \Gamma\{\vec{x}/\vec{z}\}\cdot\vec{y}:\vec{\tau}\cdot X:\vec{\alpha} \vdash_s P \triangleright \overrightarrow{\mathrm{p}y} \otimes ?^{\boldsymbol{\iota}} A\{\vec{x}/\vec{z}\} \otimes X\langle\vec{x}\rangle & (\kappa = \boldsymbol{\iota}) \\ \Gamma\{\vec{x}/\vec{z}\}\cdot\vec{y}:\vec{\tau}\cdot X:\vec{\alpha} \vdash_s P \triangleright \overrightarrow{\mathrm{p}y} \otimes ?A\{\vec{x}/\vec{z}\} \otimes X\langle x\vec{w}\rangle & (\kappa = \boldsymbol{\mu},\ s = s') \end{cases}}{\Gamma \vdash_s (\mu X(\vec{x}:\vec{\alpha}).x_0(\vec{y}:\vec{\tau}).P)\langle\vec{z}\rangle \triangleright !z_0 \otimes A}$$

In this variant, we do not care the level of the recursive subject provided it is innocent. In the same way, we can refine (Bra$^{!}$) as follows.

$$\frac{\textsf{(Bra}^{!}\textsf{-variant)}\quad \vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s,\kappa_1\&\kappa_1} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i) \qquad (j = 1, 2) \quad \begin{cases} \Gamma\{\vec{x}/\vec{z}\}\cdot\vec{y}_j:\vec{\tau}_j\cdot X:\vec{\alpha} \vdash_{s_j} P_j \triangleright \overrightarrow{\mathrm{p}y_j} \otimes ?^{\boldsymbol{\iota}} A_j\{\vec{x}/\vec{z}\} \otimes X\langle\vec{x}\rangle & (\kappa_j = \boldsymbol{\iota}) \\ \Gamma\{\vec{x}/\vec{z}\}\cdot\vec{y}_j:\vec{\tau}_j\cdot X:\vec{\alpha} \vdash_{s_j} P_j \triangleright \overrightarrow{\mathrm{p}y_j} \otimes ?A_j\{\vec{x}/\vec{z}\} \otimes X\langle x_0\vec{w}_j\rangle & (\kappa_j = \boldsymbol{\mu},\ s_j = s) \end{cases}}{\Gamma \vdash_{s_1 \sqcap s_2} (\mu X(\vec{x}:\vec{\alpha}).x_0[(\vec{y}_1:\vec{\tau}_1).P_1 \,\&\, (\vec{y}_2:\vec{\tau}_2).P_2])\langle\vec{z}\rangle \triangleright !z_0 \otimes (A_1 \odot A_2)}$$

In the conclusion we take the meet of s_1 and s_2 since they can differ when one branch is innocent and the other is not (if both are innocent or non-innocent, we can adjust their level by degradation). In such a case, the innocent branch can have a level $s_1 \leq s$, while the non-innocent branch should have a level s. As a whole, this process has a tampering level $s_1 \sqcap s_2 = s_1 \sqcap s = s_1$.

4.4. Examples of Typing

We offer a few examples of typed terms. We often use the notation $\overline{\alpha}$, which is the dual type of α, as given in Section 3.2. The abbreviations for types and processes are gathered in figure 2, where we annotate free names with their security levels, and processes with tampering levels.

Non-linearity. This example and the next are concerned with the CCS term $\overline{a}.b$ discussed in the beginning of section 2.

Let $\mathtt{sync}^{\Downarrow}_s \overset{\text{def}}{=} ()^{\Downarrow}_s$ be the type of a name used solely for nondeterministic input synchronization (hence carrying no communication) at security level s. Its dual, $\mathtt{sync}^{\Uparrow}_s$, is $()^{\Uparrow}_s$. If we assign $\mathtt{sync}^{\Uparrow}_s$ to a, and $\mathtt{sync}^{\Downarrow}_{s''}$ to b, then we have seen that secure information flow is obtained only when $s \leq s''$. In fact we can type the process $\overline{a}.b$ at security level s' somewhere

Type

$$\mathtt{sync}^{\Downarrow}_{s} \overset{\text{def}}{=} ()^{\Downarrow}_{s} \qquad \mathtt{bool}^{\uparrow}_{s} \overset{\text{def}}{=} [\ \oplus\ ]^{\uparrow}_{s} \qquad \mathtt{var}^{!}_{s} \overset{\text{def}}{=} [(\mathtt{bool}^{!}_{s})^{\uparrow}_{s} \& \mathtt{bool}^{?}_{s}]^{!}_{s,\boldsymbol{\iota}\&\boldsymbol{\mu}}$$

$$\mathtt{sync}^{\downarrow}_{s} \overset{\text{def}}{=} ()^{\downarrow}_{s} \qquad \mathtt{bool}^{\downarrow}_{s} \overset{\text{def}}{=} [\ \&\ ]^{\downarrow}_{s} \qquad \mathtt{var}^{?}_{s} \overset{\text{def}}{=} [(\mathtt{bool}^{?}_{s})^{\downarrow}_{s} \oplus \mathtt{bool}^{!}_{s}]^{?}_{s,\boldsymbol{\iota}\oplus\boldsymbol{\mu}}$$

$$\mathtt{bool}^{!}_{s} \overset{\text{def}}{=} (\mathtt{bool}^{\uparrow}_{s})^{!}_{\boldsymbol{\iota},s} \qquad \mathtt{var}_{s} \overset{\text{def}}{=} \langle \mathtt{var}^{!}_{s}, \mathtt{var}^{?}_{s} \rangle$$

$$\mathtt{bool}^{?}_{s} \overset{\text{def}}{=} (\mathtt{bool}^{\downarrow}_{s})^{?}_{\boldsymbol{\iota},s}$$

Agents ($s'' \leq s' \leq s$ in all cases)

$$\begin{aligned}
\mathbf{T}\langle b^{s} \rangle &\overset{\text{def}}{=} b(c\!:\!\mathtt{bool}^{\uparrow}_{s}).(\overline{c}\,\mathtt{inl} \mid \mathbf{T}\langle b \rangle) \\
\mathbf{If}\langle b^{s'}, P^{s}, Q^{s} \rangle &\overset{\text{def}}{=} \overline{b}(\boldsymbol{\nu}\, c\!:\!\mathtt{bool}^{\downarrow}_{s'}).c[().P \,\&\, ().Q] \\
[b^{s} \leftarrow b'^{s'}] &= b(c : \mathtt{bool}^{\uparrow}_{s}).(\mathbf{If}\langle b', \overline{c}\,\mathtt{inl}, \overline{c}\,\mathtt{inr} \rangle \mid [b \leftarrow b']) \\
\mathbf{Var}\langle x^{s} b^{s'} \rangle &= x[(z\!:\!(\mathtt{bool}^{!}_{s})^{\uparrow}_{s}).(\overline{z}(\boldsymbol{\nu}\, b'\!:\!\mathtt{bool}^{?}_{s}).[b' \leftarrow b] | \mathbf{Var}\langle xb \rangle) \& (b'\!:\!\mathtt{bool}^{?}_{s}).\mathbf{Var}\langle xb' \rangle] \\
\mathbf{Read}\langle x^{s'}, b^{s''}, P^{s} \rangle &\overset{\text{def}}{=} \overline{x}\mathtt{inl}(z\!:\!(\mathtt{bool}^{?}_{s'})^{\downarrow}_{s'}).z(b\!:\!\mathtt{bool}^{?}_{s}).P \\
\mathbf{Write}\langle b^{s'}, x^{s}, P^{s} \rangle &\overset{\text{def}}{=} \overline{x}\mathtt{inr}(b'\!:\!\mathtt{bool}^{!}_{s}).([b' \leftarrow b] \mid P) \\
\mathbf{Assign}\langle x^{s}, y^{s'}, P^{s} \rangle &\overset{\text{def}}{=} \mathbf{Read}\langle y, b, \mathbf{Write}\langle b, x, P \rangle \rangle
\end{aligned}$$

Figure 2. Useful process and type abbreviations

below both s and s''. Putting $s' \leq s \leq s''$, a simple deduction, using rules (Zero), (Out), (In), (Deg$_s$) and (Weak-$\Updownarrow$), shows that:

$$a\!:\!\mathtt{sync}^{\Uparrow}_{s} \cdot b\!:\!\mathtt{sync}^{\Downarrow}_{s''} \vdash_{s'} \overline{a}.b \triangleright \Updownarrow a \otimes \Updownarrow b$$

True linearity. If instead we decide that types for a and b ought to be linear, we can put $\mathtt{sync}^{\downarrow}_{s} \overset{\text{def}}{=} ()^{\downarrow}_{s}$, and its dual $\mathtt{sync}^{\uparrow}_{s} \overset{\text{def}}{=} ()^{\uparrow}_{s}$. In this case we may have $a\!:\!\mathtt{sync}^{\uparrow}_{s}$ and $b\!:\!\mathtt{sync}^{\downarrow}_{s'}$ for arbitrary s and s', since linear types neglect security levels. Using rules Zero, Out$^{\downarrow}$, and In$^{\uparrow}$, we can type the process at the $\top$ security level.

$$a\!:\!\mathtt{sync}^{\uparrow}_{s} \cdot b\!:\!\mathtt{sync}^{\downarrow}_{s'} \vdash_{\top} \overline{a}.b \triangleright \uparrow a \rightarrow \downarrow b$$

Note that causality is now recorded in the action type.

Branching. Boolean constants were introduced in subsection 2.2. Let $\mathtt{bool}^{\uparrow}_{s} \overset{\text{def}}{=} [\ \oplus\ ]^{\uparrow}_{s}$ be the type of a boolean value. Then, using rule Zero followed by Sel$^{\uparrow}_{r}$, we conclude $c : \mathtt{bool}^{\uparrow}_{s} \vdash_{s} \overline{c}\mathtt{inl} \triangleright \uparrow c$. For $\mathbf{T}\langle b \rangle$, let $\mathtt{bool}^{!}_{s} \overset{\text{def}}{=} (\mathtt{bool}^{\uparrow}_{s})^{!}_{s}$ be the type of a boolean constant. Starting from the above result, and using rules Var$^{!}$, Par, and In$^{!}$, we can show that

$$b\!:\!\mathtt{bool}^{!}_{s} \vdash_{s} \mathbf{T}\langle b \rangle \triangleright \mathbf{!}b$$

Let $\mathtt{bool}^{?}_{s}$ be the type of a innocuous name interacting with a boolean constant, that is, $(\mathtt{bool}^{\downarrow}_{s})^{?}_{s,\boldsymbol{\iota}}$. For the conditional $\mathbf{If}\langle b^{s'}, P^{s}_{1}, P^{s}_{2} \rangle$ introduced in the same subsection, suppose that the two branches P_1 and P_2 can be typed at a security level above that of the boolean constant b; that is, P_i is such that $\Gamma \cdot b\!:\!\mathtt{bool}^{?}_{s'} \vdash_{s} P_i \triangleright \mathbf{?}A \otimes \mathbf{?}^{\boldsymbol{\iota}} b$ and $s' \leq s$. Then we can show that (see appendix E.2.1 for details):

$$\Gamma \cdot b\!:\!\mathtt{bool}^{?}_{s'} \vdash_{s} \mathbf{If}\langle b, P_1, P_2 \rangle \triangleright A \otimes \mathbf{?}^{\boldsymbol{\iota}} b.$$

The innocuousness of b is used in rule $\mathrm{Out}^{?}$ to show that $(\mathtt{bool}^{!}_{s'})^{?}_{\top} \leq (\mathtt{bool}^{!}_{s'})^{?}_{s'}$. For b whose type is non-innocuous we would not be able to show that $(\mathtt{bool}^{!}_{s'})^{?}_{s} \leq (\mathtt{bool}^{!}_{s'})^{?}_{s'}$.

Copy-cat. The following agent concisely represents the idea of safe information flow in the present calculus. It also serves as a substitute for free name passing for various purposes, including the imperative variable and the writing of variables (see below).

$$[b^{s} \leftarrow b'^{s'}]^{s} \;=\; b(c : \mathtt{bool}^{\uparrow}_{s}).(\mathbf{If}\langle b', \overline{c}\,\mathtt{inl}, \overline{c}\,\mathtt{inr}\rangle \mid [b \leftarrow b'])$$

This agent transforms a boolean behaviour from b' to b. If $s' \leq s$ we can show that

$$b : \mathtt{bool}^{!}_{s}, b' : \mathtt{bool}^{?}_{s'} \vdash_{s} [b \leftarrow b'] \triangleright \mathbf{!}b \otimes \mathbf{?}^{\iota} b'.$$

Innocuousness at b', and the restriction $s' \leq s$ are important for the conditional (see appendix E.2.2 for details).

Imperative variable. We give a representation of an imperative variable, alternative to that presented in section 2.2.

$$\mathbf{Var}\langle x^{s} b^{s'}\rangle^{s} = x[(z : (\mathtt{bool}^{!}_{s})^{\uparrow}_{s}).(\overline{z}(\boldsymbol{\nu}\, b' : \mathtt{bool}^{?}_{s}).[b' \leftarrow b] | \mathbf{Var}\langle xb\rangle) \;\&\; (b' : \mathtt{bool}^{?}_{s}).\mathbf{Var}\langle xb'\rangle]$$

By the copy-cat [6], sending a new b' has the same effect as sending b. If $s' \leq s$, we can show that:

$$x : \mathtt{var}^{!}_{s},\ b : \mathtt{bool}^{!}_{s'} \vdash_{s} \mathbf{Var}\langle x^{s} b\rangle \triangleright \mathbf{!}x \otimes \mathbf{?}^{\iota} b.$$

Note b has the level s' but the secrecy index is still s, since at b the output is innocuous (see appendix E.2.3 for details).

Reading a variable. To better understand how an imperative variable works, we read a variable x and instantiate its value under name b in some process P.

$$\mathbf{Read}\langle x^{s'}, b^{s''}, P^{s}\rangle \stackrel{\mathrm{def}}{=} \overline{x}\mathtt{inl}(z : (\mathtt{bool}^{?}_{s'})^{\downarrow}_{s'}).z(b : \mathtt{bool}^{?}_{s}).P$$

We start by invoking the left branch of x with fresh name z and wait for the reply at this name. The reply carries a name representing the boolean value stored at the variable. If $s'' \leq s' \leq s$ and $\Gamma \cdot x : \mathtt{var}^{!}_{s'} \cdot b : \mathtt{bool}^{?}_{s''} \vdash_{s} P \triangleright \mathbf{?}x \otimes \mathbf{?}^{\iota} b$, then we can show that

$$\Gamma \cdot x : \mathtt{var}^{!}_{s'} \vdash_{s} \mathbf{Read}\langle x, b, P\rangle \triangleright \mathbf{?}x.$$

The derivation is simple: apply to the hypothesis rule $\mathrm{In}^{\downarrow}$ followed by $\mathrm{Sel}^{?}_{l}$ (see appendix E.2.4 for details). The antecedent for rule $\mathrm{In}^{\downarrow}$ requires that $(\mathtt{bool}^{?}_{s''})^{\downarrow}_{s} \leq (\mathtt{bool}^{?}_{s'})^{\downarrow}_{s'}$, thus setting up the hierarchy of security levels, $s'' \leq s' \leq s$. The security level of the boolean constant, s'', is the lowest (in fact, in section 6 boolean constants have no secrecy at all, that is are of level $\perp$). The subtyping precondition further allows the process $z(b : \mathtt{bool}^{?}_{s}).P$ to be typed at a level higher than that of x itself. Crucial for $\mathbf{Read}\langle x, b, P\rangle$ to be typed at the higher level is the fact that the left branch of the type for variables, $\mathtt{var}^{!}_{s}$, is innocuous, otherwise we wouldn't be able to check the subtyping precondition for rule $\mathrm{Sel}^{?}_{l}$.

Writing on a variable. To write the value b on a variable x and go on with process P, we use the following process, where $s' \leq s$.

$$\mathbf{Write}\langle b^{s'}, x^{s}, P^{s}\rangle \stackrel{\mathrm{def}}{=} \overline{x}\mathtt{inr}(b' : \mathtt{bool}^{!}_{s}).([b' \leftarrow b] \mid P)$$

This time we invoke the right branch of the variable with a fresh name b'. The copy-cat then links b to b', thus simulating writing b in x. If $s' \leq s$ and $\Gamma \cdot x : \mathtt{var}^{?}_{s} \cdot b : \mathtt{bool}^{?}_{s'} \vdash_s P \triangleright \mathbf{?}x \otimes \mathbf{?}^{\iota} b$, then we can show that

$$\Gamma \cdot x : \mathtt{var}^{?}_{s} \cdot b : \mathtt{bool}^{?}_{s'} \vdash_s \mathbf{Write}\langle b, x, P \rangle \triangleright \mathbf{?}x.$$

The derivation comprises two branches, one starting with copy-cat, the other with the hypothesis. Rule Par joins the branches; rule $\mathrm{Sel}^{?}_{r}$ completes the tree (see appendix E.2.5 for details). The copy-cat $[b'^{s} \leftarrow b^{s'}]^{s}$ sets up the hierarchy levels: boolean value b as the lowest security level, the security level of the copy-cat is the that of the fresh name b'. Since the right branch of the type for variables, $\mathtt{var}^{!}_{s}$, is non-innocuous, the subtyping precondition for rule $\mathrm{Sel}^{?}_{r}$ requires that the security level of $\mathbf{Write}\langle b, x, P \rangle$ is that of x, hence that of b'.

Assignment. The following offers the typing of the behaviour representing $x^{\mathrm{H}} := y^{\mathrm{L}}$, which is the prime example for innocuousness in section 2.3. To assign y to x, we read from y and write in x, using the two abbreviations above.

$$\mathbf{Assign}\langle x^{s}, y^{s'}, P^{s} \rangle \stackrel{\mathrm{def}}{=} \mathbf{Read}\langle y, b, \mathbf{Write}\langle b, x, P \rangle \rangle$$

If $s' \leq s$ and $\Gamma \cdot x : \mathtt{var}^{?}_{s} \cdot y : \mathtt{bool}^{!}_{s'} \vdash_s P \triangleright \mathbf{?}x \otimes \mathbf{?}^{\iota} y$, then we can show that

$$\Gamma \cdot x : \mathtt{var}^{?}_{s} \cdot y : \mathtt{bool}^{!}_{s'} \vdash_s \mathbf{Assign}\langle x, y, P \rangle \triangleright \mathbf{?}x \otimes \mathbf{?}y.$$

Notice that the read operation yields a process of a security level higher than that of y, but the write part requires this level to be that of x.

5. Elementary Properties of Typed Processes

5.1. Basic Syntactic Properties

This section presents basic syntactic properties of typed terms, including the subject reduction. They give basic consistency properties of the typing system. We also outline essential behavioural properties of typed terms through informal discussion, articulating what it means in general for typed processes to have safe information flow.

The first proposition below says that, if $\Gamma \vdash_s P \triangleright A$, then A is always consistent with Γ. The proof is mechanical by checking rules in Figures 1/2.

Proposition 5.1 (well-formedness of action types) *Suppose* $\Gamma \vdash_s P \triangleright A$. *Then we have:* (i) *For each* $\mathsf{p}x$ *in* A, *there exists some* $\alpha \leq \Gamma(x)$, *such that either* (a) $\mathsf{p} = \mathsf{md}(\alpha)$, (b) $\mathsf{p} = \mathbf{?}^{\iota}$ *with* $\mathsf{md}(\alpha) = \mathbf{?}$ *or* (c) $\mathsf{p} = \Updownarrow$ *with* α *non-linear, and* (ii) *For each* $X\langle \vec{x} \rangle$ *in* A, $\Gamma(X) = \vec{\alpha}$ *and* $\alpha_i \leq \Gamma(x_i)$.

The following result says that every typable term has a canonical typing, i.e. whenever P is typable under Γ, P has the minimum action type and the highest secrecy index. For the proof, we introduce an alternative typing system $\Gamma \Vdash_s P \triangleright A$ which deduces the canonical type of a given term P and Γ if it is ever typable, where we delete rules `(Deg)`, `(Weak-↕)`, `(Weak-?)` and `(Weak-⇕)` from the original system. This alternative system is presented in Appendix D. Subsequently we use the following notation:

$$A \leq_{\mathtt{G}} A' \quad \text{iff} \quad A = A'_0 \otimes \overrightarrow{\mathbf{?}^{\iota} x} \text{ and } A' = A'_0 \otimes \overrightarrow{\mathbf{?}x} \otimes \overrightarrow{\updownarrow y} \otimes \overrightarrow{\Updownarrow w}.$$

$A \leq_{\mathtt{G}} A'$ means A' is an action type extending A'. We can now present the basic properties of the canonical typing systems.

Proposition 5.2 (1) ($\Vdash$ is $\vdash$) $\Gamma \Vdash_s P \triangleright A$ *implies* $\Gamma \vdash_s P \triangleright A$

(2) (the uniqueness of $\Vdash$) $\Gamma \Vdash_{s_1} P \triangleright A_1$ *and* $\Gamma \Vdash_{s_2} P \triangleright A_2$ *imply* $s_1 = s_2$ *and* $A_1 = A_2$.

(3) ($\Vdash$ has smaller index and type) $\Gamma \Vdash_s P \triangleright A$ *and* $\Gamma \vdash_{s'} P \triangleright A'$ *imply* $s' \leq s$ *and* $A \leq_{\mathtt{G}} A'$.

(4) (canonical typing) *If* $\Gamma \vdash_s P \triangleright A$*, then there exists* s_0 *and* A_0 *such that* (i) $\Gamma \vdash_{s_0} P \triangleright A_0$ *and* (ii) *whenever* $\Gamma \vdash_{s_1} P \triangleright A_1$ *we have* $s_1 \leq s_0$ *and* $A_0 \leq_{\mathtt{G}} A_1$.

Proof: Mechanical by rule induction. ■

Using the canonical typing system, we establish the standard properties of the original typing system. Note the subsumption property holds for channel types even if they are placed in the base: this is consistent with our intuition that, in $\Gamma \vdash_s P \triangleright A$, channel types in Γ represent the constraints on the behaviour of P, rather than that of the outside environment.

Lemma 5.3

(1) (closure under α-conversion and renaming) *Let* $P \equiv_\alpha Q$ *and* $\mathsf{fn}(Q) \cap \mathsf{bn}(\Gamma) = \emptyset$*. Then* $\Gamma \vdash_s P \triangleright A$ *implies* $\Gamma \vdash_s Q \triangleright A$*. Also, for any injective name substitution* σ *and assuming bound names are appropriately chosen,* $\Gamma\sigma \vdash_s P\sigma \triangleright A\sigma$*. Similarly for agent variables.*

(2) (subsumption) *If* $\Gamma \cdot x{:}\alpha \vdash_s P \triangleright A$ *and* $\vdash \alpha \leq \alpha'$*, then* $\Gamma \cdot x{:}\alpha' \vdash_s P \triangleright A$.

(3) (narrowing) *If* $\Gamma \cdot X{:}\vec{\alpha} \vdash_s P \triangleright A$ *and* $\vdash \alpha_i \geq \beta_i$ *for each* i *then* $\Gamma \cdot X{:}\vec{\beta} \vdash_s P \triangleright A$.

(4) (weakening) *If* $\Gamma \vdash_s P \triangleright A$ *and* $x \notin \mathsf{fn}(\Gamma)$*, then for any well-formed* α *we have* $\Gamma \cdot x : \alpha \vdash_s P \triangleright A$*. Similarly if* $\Gamma \vdash_s P \triangleright A$ *and* $X \notin \mathsf{fv}(\Gamma)$*, then for any well-formed* $\vec{\alpha}$ *we have* $\Gamma \cdot X{:}\vec{\alpha} \vdash_s P \triangleright A$.

(5) (strengthening) *If* $\Gamma \cdot x : \alpha \vdash_s P \triangleright A$ *and* $x \notin \mathsf{fn}(P)$*, then* $\Gamma \vdash_s P \triangleright A/x$*. Similarly if* $\Gamma \cdot X{:}\vec{\alpha} \vdash_s P \triangleright A$ *and* $X \notin \mathsf{fv}(P)$*, then* $\Gamma \vdash_s P \triangleright A$.

Proof: See Appendix E.3.1. ■

The substitution lemma concerning recursions follows. Notice the difference between innocuous recursion and non-innocuous recursion. The recursive behaviour at the non-innocuous channel changes the action types, while one at the innocuous channel does not effect anything.

Lemma 5.4 (variable substitution)

(1) (non-linear) *Suppose* $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright A$ *with* $\mathcal{E} \overset{def}{=} \mu X(\vec{x}).P$ *and* $\Gamma(x_0)$ *non-linear. Then:*
$\Gamma \cdot X{:}\vec{\alpha} \vdash_s P \triangleright A$ *implies* $\Gamma \vdash_s P\{\mathcal{E}/X\} \triangleright A$.

(2) ($\mathrm{in}^{!}_{\boldsymbol{\mu}}$) *If* $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright \mathbf{?}A \otimes \mathbf{!}x_0$ *with* $\mathcal{E} \overset{def}{=} \mu X(\vec{x}).x_0(\vec{y}{:}\tau).P$ *and* $\vdash (\vec{\tau})^{!}_{s,\boldsymbol{\iota}} \leq \Gamma(x_0)$*, then* $\Gamma \cdot \vec{y}{:}\vec{\tau} \cdot X{:}\vec{\alpha} \vdash_s P \triangleright \overrightarrow{\mathtt{p}y} \otimes \mathbf{?}A \otimes X\langle x_0\vec{w}\rangle$ *implies* $\Gamma \cdot \vec{y}{:}\vec{\tau} \vdash_s P\{\mathcal{E}/X\} \triangleright \overrightarrow{\mathtt{p}'y} \otimes \mathbf{?}A \otimes \mathbf{!}x_0$
where $\mathtt{p}'_i = \mathtt{p}_i$ *or* $\mathtt{p}'_i = \mathbf{?}$ *with* $\mathtt{p}_i = \mathbf{?}^{\boldsymbol{\iota}}$.

(3) ($\mathrm{bra}^{!}_{\boldsymbol{\mu}}$) *If* $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright \mathbf{?}A_1 \odot \mathbf{?}A_2 \otimes \mathbf{!}x_0$ *with* $\mathcal{E} \overset{def}{=} \mu X(\vec{x}).x_0[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2]$ *and* $\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s,\kappa_1\&\kappa_2} \leq \Gamma(x_0)$ *with* $\kappa_j = \boldsymbol{\mu}$ $(j = 1$ *or* $2)$*, then* $\Gamma \cdot \vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \vdash_{s_j} P \triangleright \overrightarrow{\mathtt{p}y_j} \otimes \mathbf{?}A_j \otimes X\langle x_0\vec{w}\rangle$ *implies* $\Gamma \cdot \vec{y}_1{:}\vec{\tau}_1 \vdash_s P\{\mathcal{E}/X\} \triangleright \overrightarrow{\mathtt{p}'y_j} \otimes \mathbf{?}A_j \otimes \mathbf{!}x_0$
where $\mathtt{p}'_{ji} = \mathtt{p}_{ji}$ *or* $\mathtt{p}'_{ji} = \mathbf{?}$ *with* $\mathtt{p}'_{ji} = \mathbf{?}^{\boldsymbol{\iota}}$ *with* $s \leq s_j$.

(4) ($\mathrm{in}^{!}_{\boldsymbol{\iota}}$) *If* $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright \mathbf{?}^{\boldsymbol{\iota}}A \otimes \mathbf{!}x_0$ *with* $\mathcal{E} \overset{def}{=} \mu X(\vec{x}).x_0(\vec{y}{:}\tau).P$ *and* $\vdash (\vec{\tau})^{!}_{s,\boldsymbol{\iota}} \leq \Gamma(x_0)$*, then* $\Gamma \cdot \vec{y}{:}\vec{\tau} \cdot X{:}\vec{\alpha} \vdash_s P \triangleright \overrightarrow{\mathtt{p}y} \otimes \mathbf{?}^{\boldsymbol{\iota}}A \otimes X\langle\vec{x}\rangle$ *implies* $\Gamma \cdot \vec{y}{:}\vec{\tau} \vdash_s P\{\mathcal{E}/X\} \triangleright \overrightarrow{\mathtt{p}y} \otimes \mathbf{?}^{\boldsymbol{\iota}}A \otimes \mathbf{!}x_0$.

(5) ($\mathrm{bra}^{!}_{\boldsymbol{\iota}}$) *Suppose* $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright \mathbf{?}^{\boldsymbol{\iota}}A_1 \odot \mathbf{?}A_2 \otimes \mathbf{!}x_0$ *with* $\mathcal{E} \overset{def}{=} \mu X(\vec{x}).x_0[(\vec{y}_1 : \vec{\tau}_1).P_1 \& (\vec{y}_2 : \vec{\tau}_2).P_2]$ *and* $\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s,\kappa_1\&\kappa_2} \leq \Gamma(x_0)$ *with* $\kappa_j = \boldsymbol{\iota}$ $(j = 1$ *or* $2)$*. Then:*
$\Gamma \cdot \vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \vdash_{s_j} P \triangleright \overrightarrow{\mathtt{p}y_j} \otimes \mathbf{?}^{\boldsymbol{\iota}}A_j \otimes X\langle\vec{x}\rangle$ *implies* $\Gamma \cdot \vec{y}_j : \vec{\tau}_j \vdash_s P\{\mathcal{E}/X\} \triangleright \overrightarrow{\mathtt{p}y_j} \otimes \mathbf{?}^{\boldsymbol{\iota}}A_j \otimes \mathbf{!}x_0$
with $s \leq s_j$.

PROOF: See Appendix E.3.2. ■

For the closure under the structual rules, algebraic properties on action types play an essential role.

Proposition 5.5 (closure under $\equiv$) *If* $\Gamma \vdash_s P \triangleright A$ *and* $P \equiv Q$ *then* $\Gamma \vdash_s Q \triangleright A$.

PROOF: See Appendix E.3.3. ■

The following theorem says that whatever internal reduction takes place, its composability with the outside, which is controlled by both Γ and A, does not change; and that, moreover, the process is still secure with a no less secrecy index. Below we let $\twoheadrightarrow \overset{\text{def}}{=} (\equiv \cup \longrightarrow)^*$, where $\longrightarrow$ and $\equiv$ are the reduction and the structural congruence on preterms defined as those for untyped terms (see Appendix B for the formal definitions).

Theorem 5.6 (subject reduction)

$$\textit{If } \Gamma \vdash_s P \triangleright A \textit{ and } P \twoheadrightarrow P' \textit{ with } \mathsf{bn}(P') \cap \mathsf{fn}(\Gamma) = \emptyset\textit{, then } \Gamma \vdash_s P' \triangleright A.$$

PROOF: By Proposition 5.5 we only have to consider one step reduction between two prefixed terms. We use the minimum typing system in Appendix D and show:

$$\begin{gathered}\text{If } \Gamma \Vdash_s P \triangleright A \text{ and } P \longrightarrow P' \text{ with } \mathsf{bn}(P') \cap \mathsf{fn}(\Gamma) = \emptyset, \\ \text{then } \Gamma \Vdash_{s'} P' \triangleright A' \text{ with } s \leq s' \text{ and } A' \leq_{\mathtt{G}} A.\end{gathered}$$

This implies the theorem by Proposition 5.2 (4). Note that the minimum subtypining system also satisfies the basic properties stated in Lemma 5.3 (1,4,5), Lemma 5.4 and Proposition 5.5. For (subsumption-narrowing) in Lemma 5.3, we assume $\mathsf{sec}(\alpha) = \mathsf{sec}(\alpha')$. See Appendix E.3.4 for detail. ■

The subject reduction is the basis of various significant behavioural properties for processes, which include, among others, safe information flow. We conclude this subsection by basic observations on secrecy levels of well-typed terms. It is useful to introduce two, mutually dual, basic classifications of single types.

Definition 5.7 Suppose P is typable under Γ. Then we say P under Γ *tampers* x *at level* s iff $P \twoheadrightarrow P' \mid R$ for some P' such that either:

(i) P' is input-prefixed (including recursive input) with the initial free name x for which $\Gamma(x)$ is nonlinear, or

(ii) P' is output-prefixed with the initial free name x and $\Gamma(x)$ is either $\alpha_{\mathtt{O}}$ or $\langle \alpha_{\mathtt{I}},\ \alpha_{\mathtt{O}} \rangle$ s.t. one of the following holds: (1) $\alpha_{\mathtt{O}}$ is nonlinear, (2) $\alpha_{\mathtt{O}}$ is truly linear but not unary, (3) $\alpha_{\mathtt{O}} = (\vec{\tau})^{?}_{s,\boldsymbol{\mu}}$, (4) $\alpha_{\mathtt{O}} = [\vec{\tau}_1 \oplus \vec{\tau}_2]^{?}_{s,\kappa_1 \oplus \kappa_2}$ and $P' \equiv x\mathtt{inl}(\boldsymbol{\nu}\,\vec{y} : \vec{\tau}).P''$ and $\kappa_1 = \boldsymbol{\mu}$, or (5) $\alpha_{\mathtt{O}} = [\vec{\tau}_1 \oplus \vec{\tau}_2]^{?}_{s',\kappa_1 \oplus \kappa_2}$ and $P' \equiv x\mathtt{inr}(\boldsymbol{\nu}\,\vec{y} : \vec{\tau}).P''$ and $\kappa_2 = \boldsymbol{\mu}$.

If P under Γ tampers x at level s for some x then we simply say P under Γ *tampers at* s.

Thus "P tampers at s" says that P does an action at a free name which is either: (a) a nonlinear input/output, (b) a truly-linear selection, or (c) a non-innocuous **?**-moded output. We can then show:

Proposition 5.8 (honest tampering) Suppose $\Gamma \vdash_s P \triangleright A$ and P under Γ tampers at s'. Then $s' \geq s$.

PROOF: It is easy by inspecting each prefix rule for P' in the definition of tampering. We then use Subject Reduction (theorem 5.6). ■

The honest tampering says that, as far as we only consider the actions of kinds (a) (b) and (c) above induce information flow, which can indeed be justified behaviourally, the derived sequent $\Gamma \vdash_s P \triangleright A$ does gurantee that P never touches the environment below s (and indeed does so only at s or higher). Note also whenever $\Gamma \vdash_s P \triangleright A$ is derivable, all subterms of P is also typed at s or above, so that the same property holds for all subterms of P (under appropriate extensions of Γ).

5.2. Discussions: Behavioural Properties of Typed Terms

The above tamparing proposition gives one simple basis for the *non-interference property*: a process interacting at the high-level security never influences a process at the low-level security. Such properties are often best or necessarily stated using typed behavioural equivalences. In the following we informally discuss basic elements of a behavioural theory of typed processes in relationship to the notion of safe information flow, leaving its detailed technical treatment to the sequel [8] to the present paper. We note that the discussion in the present subsection is not formally used in Section 6.

We first note that behavioural properties of typed terms in the present context include (1) those which purely concern deadlock-freedom and innocence and (2) those which concern secrecy-sensitive behaviours of processes. It is important to notice that, as our informal discussion in Section 2 has suggested, (1) is a prerequisite for (2), i.e. the "safety" in the context of information flow can be properly formulated only in the context of terms typed under secrecy-insensitive typing. Therefore it makes sense to consider the behavioural content of safe information flow in the present calculus in the following two stages.

1. Firstly, we consider a larger set of terms that are typable under the typing system which differs from that of Section 4 only in that it neglects all secrecy levels. We may call such terms *behaviourally typed terms*;
2. Secondly, we stipulate what would constitues a "safe" behaviour in the context of the enlarged set of terms in (i). This gives a behavioural characterisation of safe information flow, with which the securely typed terms (i.e. terms typed under the system in Section 4) should comply.

The same articulation can be applied to behavioural equivalences: we have a basic theory of bisimilarity in the context of (i), on the basis of which a secrecy-sensitive notion of bisimilarity parameterised by secrecy levels is constructed. In the latter, the parameterisation by a secrecy level means we consider the equivalence of two behaviours from the viewpoint of an observer with a certain security level: thus, for example, two high-level outputs at different channels are indistinguishable for an observer at a low-level security. Naturally the latter equivalence equates more terms, as far as securely typed terms go (thus inducing a partial equivalence on behaviourally typed terms, cf.[45]).

The bisimilarity on behaviourally typed terms uses the labelled transition relation of form $\Gamma \vdash P \xrightarrow{l} \Delta \vdash P'$ (action types are omitted since they are again canonically inferred). Detailed technical treatment of the transition and related notions need some preparations and are beyond the scope of the present paper. In the following, we discuss behavioural properties informally, emphasising those properties whose statement need neither transition relation nor bisimilarity.

We start from behavioural properties. We can first easily show that non-trivial modalities, $!$, $\uparrow$, and $\downarrow$, in the action type of a typed term, do gurantee existence of the corresponding actions, as well as expected consequences: $!$ ensures repeated availability of input actions, while $\uparrow$ and $\downarrow$ gurantees, essentially speaking, that the corresponding input/output action is done exactly once at that channel. Further if a channel with mode $!$ is given an innocent type, it has a property that its part with the same free names as before does not change after the input action, while it may add a new term with newly imported names. These properties include both liveness and safety properties, and are important for establishing the concgruence of the induced bisimilarity.

With respect to 5.8, we note that processes which only tamper at levels less than s, as in the conclusion of the above remark, are regarded as having no significant behaviour by secrecy-sensitive behavioural equivalences at level s (in particular if its channels are all non-deterministic, it can be equated with the inaction). Thus whenever $\Gamma \vdash_s P_{1,2} \triangleright A$ and $s' \lnsim s$, we have $\Gamma \vdash_s P_1 \overset{\star}{\sim}_{s'} P_2 \triangleright A$ where $\overset{\star}{\sim}_s$ is some secrecy-sensitive behavioural equivalence. This is one of the fundamental properties which is often used in various proofs.

Secrecy-sensitive behavioural equivalences can be formulated using various notions of process equivalences [21, 23, 38]. We have so far studied two of such constructions, both based on co-inductive definitions. One is a bisimilarity (which has weak and strong versions), defined on behaviourally typed terms, and another is a basic reduction-closed congruence on securely typed terms. The bisimilarity offers the notion of safe behaviours for behaviourally typed terms in the context of information flow: for example, if x is typed as non-deterministic at level s under Γ, then if a safe process does an action at x then it should not change the behaviour of that process up to the bisimilarity at level s' which is smaller than s. The bisimilarity restricted on securely typed terms is a subcongruence of the reduction-based equality, which is fundamental for reasoning about behaviours of securely typed terms.

Leaving the technical construction of bisimilarities to the sequel, here we discuss how the reduction-based congruence can be defined, since it needs lesser machinery. Using the congruence, we state a most basic property of securely typed terms, a behavioural non-interference property. First, define the collection of (one-hole) *typed contexts* following the typing rules in Section 4. A $\langle \Gamma \cdot s \cdot A \rangle$*-context* is a typed context whose hole is typed under the triple $\langle \Gamma, s, A \rangle$. Then the notion of *typed congruence* is naturally defined: it is an equivalence relation over typed terms which respects typings and which is closed under typed contexts (by "respects typings" we mean related typed terms should own the identical base, the secrecy level and the action type).

Now let $\{\cong_s\}$ ("the family of s-sensitive barbed reduction-closed congruences") be the maximum typed congruences such that, whenever $\Gamma \vdash_{s_0} P \cong_s Q \triangleright A_0$, the following two conditions hold: (1) if $P \twoheadrightarrow P'$ then $Q \twoheadrightarrow Q'$ s.t. $\Gamma \vdash_{s_0} P' \cong_s Q' \triangleright A_0$; and (2) if A_0 is *closed*, that is if A only contains nodes of form $\mathsf{p}_i x_i$ with $\mathsf{p}_i \in \{\mathbf{!}, \Updownarrow\}$ (which says all deadlock-free actions at free channels are enabled), then, for each x s.t. $\mathsf{sec}(\Gamma(x)) \leq s$ ($\mathsf{sec}(\alpha)$ is given by the lub if α is a pair type), we have $P \Downarrow_{x^\rho} \Leftrightarrow Q \Downarrow_{x^\rho}$ for $\rho \in \{\mathtt{I}, \mathtt{O}\}$, where we define $P \Downarrow_{x^{\mathtt{I}}}$ as $P \twoheadrightarrow P' \,|\, R$ for some P' such that P' is input-prefixed with the initial free name x, dually for $P \Downarrow_{x^{\mathtt{O}}}$. We can now state the non-interference result (cf. [17, 1, 37]).

Remark 5.9 (behavioural non-interference) Let $C[\cdot]$ be a $\langle \Gamma_0 \cdot s_0 \cdot A_0 \rangle$-context. If $s \lnsim s_0$ and $\Gamma_0 \vdash_{s_0} P_i \triangleright A_0$ $(i = 1, 2)$, then $C[P_1] \cong_s C[P_2]$.

The statement says that the behaviour of the whole at lower levels is never affected by its constituting behaviours which only act at higher levels. The proof uses the secrecy-sensitive typed bisimilarity which we mentioned above. By noting ground constants are representable as constant behaviours, one may say that the result extends Abadi's non-interference result for ground values [1] to typed process behaviours.

6. Imperative Secure Information Flow as Typed Process Behaviour

6.1. A Multi-threaded Imperative Calculus

Smith and Volpano [50] presented a type discipline for a basic multi-threaded imperative calculus in which well-typedness ensures secure information flow. In this section we show how the original system can be embedded in the typed calculus introduced in this paper, with a suggestion for a practically interesting extension of the original type discipline through the analysis of the notion of observables. We start with the syntax of untyped phrases of

(Subtyping $\rho \leq \rho'$) $\dfrac{s' \leq s}{s\ \mathtt{cmd}_{\Downarrow} \leq s'\ \mathtt{cmd}_{\Downarrow}}$ $\dfrac{s' \leq s}{s\ \mathtt{cmd}_{\Downarrow} \leq s'\ \mathtt{cmd}_{\Uparrow}}$ $\dfrac{}{s\ \mathtt{cmd}_{\Uparrow} \leq s\ \mathtt{cmd}_{\Uparrow}}$

(Typing System $E \vdash e : s$)

$$(\mathtt{var})\ \frac{E(x) \leq s}{E \vdash x : s} \qquad (\mathtt{bool})\ E \vdash b : s \qquad (\mathtt{and})\ \frac{E \vdash e_i : s\ (i = 1, 2)}{E \vdash e_1\ \mathtt{and}\ e_2 : s}$$

(Typing System $E \vdash c : \rho$)

$(\mathtt{skip})\ \ E \vdash \mathtt{skip} : s\ \mathtt{cmd}_{\Downarrow}$

$$(\mathtt{subs})\ \frac{E \vdash c : \rho \quad \rho \leq \rho'}{E \vdash c : \rho'} \qquad (\mathtt{compose})\ \frac{E \vdash c_i : \rho \quad (i = 1, 2)}{E \vdash c_1 ; c_2 : \rho} \qquad (\mathtt{parallel})\ \frac{E \vdash c_i : \rho \quad (i = 1, 2)}{E \vdash c_1 \mid c_2 : \rho}$$

$$(\mathtt{assign})\ \frac{\begin{array}{l} E \vdash e : s \\ E(x) = s \end{array}}{E \vdash x := e : s\ \mathtt{cmd}_{\Downarrow}} \qquad (\mathtt{if})\ \frac{\begin{array}{l} E \vdash e : s \qquad \mathsf{sec}(\rho) = s \\ E \vdash c_i : \rho \quad (i = 1, 2) \end{array}}{E \vdash \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 : \rho} \qquad (\mathtt{while})\ \frac{\begin{array}{l} E \vdash e : \bot \\ E \vdash c : \bot\ \mathtt{cmd}_{\Uparrow} \end{array}}{E \vdash \mathtt{while}\ e\ \mathtt{do}\ c : \bot\ \mathtt{cmd}_{\Uparrow}}$$

Figure 3. Typing System of Smith-Volpano calculus

the original calculus, using $x, y, z, ...$ for imperative variables.

$$\begin{array}{lrcl} (\mathtt{value}) & \mathtt{b} & ::= & \mathtt{tt} \mid \mathtt{ff} \\ (\mathtt{expression}) & e & ::= & x \mid \mathtt{b} \mid e_1\ \mathtt{and}\ e_2 \\ (\mathtt{command}) & c & ::= & x := e \mid c_1 ; c_2 \mid c_1 \mid c_2 \mid \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ e\ \mathtt{do}\ c \mid \mathtt{skip} \end{array}$$

He have modified the original calculus in three ways: for simplicity we restrict data types to booleans,[6] for convenience we added the `skip` command, and we use the parallel composition rather than a system of threads.

The typing system for the calculus is given in Figure 3. We have changed the original type system in three ways. First, secrecy levels are taken from an arbitrary secrecy lattice rather than the two-point lattice (composed of H and L). Second, variables are assigned secrecy levels rather than arbitrary command types (variables of command type, possible in the original system, seem to be of no use [2]). Third, we make the notion of divergence explicit in the types for commands. Writing $s, s', \ldots$ for secrecy levels as before, the extended syntax of the *command types* follows.

$$\rho \quad ::= \quad s\ \mathtt{cmd}_{\Downarrow} \quad \mid \quad s\ \mathtt{cmd}_{\Uparrow}.$$

Here $s\ \mathtt{cmd}_{\Downarrow}$ (resp. $s\ \mathtt{cmd}_{\Uparrow}$) indicates convergent (resp. divergent) phrases. In Figure 3, the base E is a finite map from variables to secrecy levels. Subsumption in expressions is merged into their typing rules for simplicity. Notice the contravariance in the first two subtyping rules [53, 50] and the invariance in the last rule.

The types in the original system [50] are embedded into the command types above by

---

[6] By introducing the indexed branching or the free name passing, any standard constant types can be treated. However all key arguments stay unchanged from the present restricted setting.

(Type and Base)

$$\llbracket s\ \mathtt{cmd}{\Downarrow}\rrbracket \overset{\text{def}}{=} \mathtt{sync}_s^{\uparrow} \quad \langle\!\langle s\ \mathtt{cmd}{\Downarrow}\rangle\!\rangle_f \overset{\text{def}}{=} \uparrow f \qquad \llbracket\emptyset\rrbracket \overset{\text{def}}{=} \mathit{tt}, \mathit{ff}:\mathtt{var}_\perp \qquad \langle\!\langle\emptyset\rangle\!\rangle \overset{\text{def}}{=} \mathbf{?}^\iota \mathit{tt} \otimes \mathbf{?}^\iota \mathit{ff}$$

$$\llbracket s\ \mathtt{cmd}{\Uparrow}\rrbracket \overset{\text{def}}{=} \mathtt{sync}_s^{\Uparrow} \quad \langle\!\langle s\ \mathtt{cmd}{\Uparrow}\rangle\!\rangle_f \overset{\text{def}}{=} \Updownarrow f \quad \llbracket E\cdot x:s\rrbracket \overset{\text{def}}{=} \llbracket E\rrbracket \cdot x:\mathtt{var}_s \quad \langle\!\langle E\cdot x:s\rangle\!\rangle \overset{\text{def}}{=} \llbracket E\rrbracket \cdot \mathbf{!}x$$

(Command) $(s = \mathsf{sec}(e)_E$ in all cases, $\mathtt{var}_s \overset{\text{def}}{=} \langle \mathtt{var}_s^{\mathbf{!}}, \mathtt{var}_s^{\mathbf{?}}\rangle)$

$$\llbracket E\vdash \mathtt{skip}:\rho\rrbracket_f \overset{\text{def}}{=} \overline{f}$$

$$\llbracket E\vdash c_1;c_2:\rho\rrbracket_f \overset{\text{def}}{=} (\boldsymbol{\nu}\, g:\langle\overline{\llbracket\rho\rrbracket},\llbracket\rho\rrbracket\rangle)(\llbracket E\vdash c_1:\rho\rrbracket_g \mid g.\llbracket E\vdash c_2:\rho\rrbracket_f)$$

$$\llbracket E\vdash c_1\mid c_2:\rho\rrbracket_f \overset{\text{def}}{=} (\boldsymbol{\nu}\, f_1,f_2:\langle\overline{\llbracket\rho\rrbracket},\llbracket\rho\rrbracket\rangle)(\llbracket E\vdash c_1:\rho\rrbracket_{f_1} \mid \llbracket E\vdash c_2:\rho\rrbracket_{f_2} \mid f_1.f_2.\overline{f})$$

$$\llbracket E\vdash x:=e:\rho\rrbracket_f \overset{\text{def}}{=} \mathbf{eval}\llbracket e\rrbracket^E(b^{s'}).\overline{x}\mathtt{inr}(b':\mathtt{bool}_{s'}^{\mathbf{!}}).([b'\leftarrow b]\mid P) \qquad (s'=E(x))$$

$$\llbracket E\vdash \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2:\rho\rrbracket_f \overset{\text{def}}{=} \mathbf{eval}\llbracket e\rrbracket^E(b^s).\mathbf{If}\langle b,\llbracket E\vdash c_1:\rho\rrbracket_f,\llbracket E\vdash c_2:\rho\rrbracket_f\rangle$$

$$\llbracket E\vdash \mathtt{while}\ e\ \mathtt{do}\ c:\rho\rrbracket_f \overset{\text{def}}{=} (\boldsymbol{\nu} g:\langle\overline{\llbracket\rho\rrbracket},\llbracket\rho\rrbracket\rangle)(\overline{g}\mid \mathcal{E}\langle fg\vec{x}\rangle) \qquad (E=\{\vec{x}:\vec{s}\},\alpha_i=\mathtt{var}_{s_i})$$

$$\text{where } \mathcal{E} \overset{\text{def}}{=} \mu X(f,g:\langle\overline{\llbracket\rho\rrbracket},\llbracket\rho\rrbracket\rangle,\vec{x}:\vec{\alpha}).\ g.\mathbf{eval}\llbracket e\rrbracket^E(b^s).\mathbf{If}\langle b,(\llbracket E\vdash c:\rho\rrbracket_g \mid X\langle fg\vec{x}\rangle),\overline{f}\rangle$$

(Expression)

$$\mathbf{eval}\llbracket x\rrbracket^E(b^s).P \overset{\text{def}}{=} \overline{x}\mathtt{inl}(z:(\mathtt{bool}_{s'}^{\mathbf{?}})_{s'}^{\downarrow}).z(b:\mathtt{bool}_s^{\mathbf{?}}).P \qquad (s'=E(x))$$

$$\mathbf{eval}\llbracket \mathtt{tt}\rrbracket^E(b^s).P \overset{\text{def}}{=} \mathbf{Link}\langle b^s,\llbracket \mathtt{b}^\perp\rrbracket,P\rangle \qquad (\llbracket\mathtt{tt}\rrbracket \overset{\text{def}}{=} \mathit{tt},\quad \llbracket\mathtt{ff}\rrbracket \overset{\text{def}}{=} \mathit{ff})$$

$$\begin{aligned}\mathbf{eval}\llbracket e_1\ \mathsf{and}\ e_2\rrbracket^E(b^s).P &\overset{\text{def}}{=} \mathbf{eval}\llbracket e_1\rrbracket^E(b_1^{s_1}).\mathbf{eval}\llbracket e_2\rrbracket^E(b_2^{s_2}).\\ &\quad \mathbf{If}\langle b_1^{s_1},\mathbf{Link}\langle b^s,b_2^{s_2},P\rangle,\mathbf{Link}\langle b^s,b_1^{s_1},P\rangle\rangle \qquad (s_i=\mathsf{sec}(e_i)_E, s\geq s_1\sqcup s_2)\end{aligned}$$

$$\mathbf{Link}\langle b^s,b'^{s'},P\rangle \overset{\text{def}}{=} (\boldsymbol{\nu}\, b:\mathtt{var}_s)(P\mid[b\leftarrow b']) \qquad (s'\leq s)$$

(Security of an expression)

$$\mathsf{sec}(x)_E \overset{\text{def}}{=} E(x) \qquad \mathsf{sec}(\mathtt{b})_E \overset{\text{def}}{=} \perp \qquad \mathsf{sec}(e_1\ \mathsf{and}\ e_2)_E \overset{\text{def}}{=} \mathsf{sec}(e_1)_E \sqcup \mathsf{sec}(e_2)_E$$

Figure 4. Translation of the Smith-Volpano calculus

setting

$$(\textsc{h})^\circ \overset{\text{def}}{=} \top \qquad (\textsc{h}\ \mathtt{cmd})^\circ \overset{\text{def}}{=} \top\ \mathtt{cmd}{\Downarrow}$$
$$(\textsc{l})^\circ \overset{\text{def}}{=} \perp \qquad (\textsc{l}\ \mathtt{cmd})^\circ \overset{\text{def}}{=} \perp\ \mathtt{cmd}{\Uparrow}$$

thus making explicit the element of termination in the original types. We note the following, identifying a system of n threads (processes) with the corresponding n-fold parallel composition.

Proposition 6.1 (conservative extension) *Let η,η' be types in [50]. Then $E\vdash c:\eta$ (resp. $\eta\leq\eta'$) in [50] if and only if $E\vdash c:\eta^\circ$ (resp. $\eta^\circ\leq\eta'^\circ$) in Figure 3.*

PROOF: See Appendix E.4.1. ■

As we shall discuss later, the system in Figure 3 is quite close to the original system in that all typed terms enjoy the identical non-interference property as stated in [50].

6.2. Embedding

We start with the embedding of types and bases, which is given in (Type and Base) in Figure 4. Both command types and bases are translated into two forms, one using channel types and the other using action types. In $\llbracket \rho \rrbracket$, a terminating type becomes a truly linear synchronisation type and a non-terminating type becomes a non-linear synchronisation type, both described in Section 4.4. $\langle\!\langle \rho \rangle\!\rangle_f$ gives an action type accordingly. $\llbracket E \rrbracket$ is translated into a basis in our sense, allowing both way interactions at variables. Accordingly $\langle\!\langle E \rangle\!\rangle$ is defined. Notice that the types for the shared boolean constants (`tt` and `ff`) are incorporated in the translation of the basis E.

The original order on the command types is faithfully preserved by the embedding in the following way.

Proposition 6.2 *Let* ρ, ρ' *be command types. Then* $\rho \leq \rho'$ *if and only if either* (1) $\mathsf{sec}(\llbracket \rho \rrbracket) \geq \mathsf{sec}(\llbracket \rho' \rrbracket)$ *and both are truly linear unary,* (2) $\mathsf{sec}(\llbracket \rho \rrbracket) \geq \mathsf{sec}(\llbracket \rho' \rrbracket)$*,* $\llbracket \rho \rrbracket$ *is truly linear unary and* $\llbracket \rho' \rrbracket$ *is nonlinear, or* (3) $\llbracket \rho \rrbracket = \llbracket \rho' \rrbracket$ *and both are nonlinear.*

PROOF: See Appendix E.4.2. ■

Note the secrecy ordering in (1) is consistent with subsumption since truly linear unary types do not care secrecy levels. The above logical equivalence may be understood as dissecting command types into (a) the secrecy level of the whole behaviour (which guarantees the lowest tampering level and which can be degraded by the degradation rule) and (b) the nature of the termination behaviour (noting "non-linear" means a termination action is not guaranteed).

We next turn to the embedding of terms into processes, given in Figure 4. The framework assumes two boolean constant agents whose behaviours are given in Section 2.2 and which are shared by all processes, with principal channels *tt* and *ff*. These free channels are given the $\bot$-level, which is in accordance with Smith and Volpano's idea that constants have no secrecy.[7]

In accordance with the translation of types, each command becomes a process which, when it terminates, emits an output signal at a channel given as a parameter, typically f (cf. [6, 38]). We are using copy-cat in Section 4.4 to represent the functionality of value passing. One complication arises concerning the assignment, conditional and while-loop, in their use of the *evaluation agent* $\mathbf{eval}\llbracket e \rrbracket^E(b^s).P$. This agent interacts with variables and constants, names the result as b with security level s, and uses it in P (we may easily extend the encoding of `and` to other operators). The translation of expressions is in the (Expression) part of the figure. If expression e is typable with the Smith-Volpano system in Figure 3, and process P can be typed at security level s, assuming a boolean constant b of a lower security level s', then $\mathbf{eval}\llbracket e \rrbracket^E(b^{s'})$ is typable as well.

Lemma 6.3 (Eval) *If* $E \vdash e : s$*, and* $\Gamma \cdot \llbracket E \rrbracket \cdot b{:}\mathtt{bool}^{?}_{s'} \vdash_s P \triangleright \mathbf{?}A \otimes \langle\!\langle \tau \rangle\!\rangle_f \otimes \mathbf{?}^{\iota} b$ *where* $s' \leq s$ *and* $\overline{\langle\!\langle E \rangle\!\rangle} \subseteq A$*, then* $\Gamma \cdot \llbracket E \rrbracket \vdash_s \mathbf{eval}\llbracket e \rrbracket^E(b^{s'}).P \triangleright A \otimes \langle\!\langle \tau \rangle\!\rangle_f$.

PROOF: The proof is by induction on the length of the derivation $E \vdash e : s$, and uses the results for **Read** and **Link** (propositions E.4 and E.7). See Appendix E.4.4 for details. ■

The encoding of terms should be easily understandable, following the known treatment as in [38]: the interest however lies in how *typability* is transformed via the embedding, and how this transformation sheds light on safe information-flow in the original system.

The key result concerning the typability says that typability in Smith-Volpano system implies the typability of the embedded term in our system.

Theorem 6.4 (Soundness) *If* $E \vdash c : \rho$*, then* $\llbracket E \rrbracket \cdot f : \llbracket \rho \rrbracket \vdash_s \llbracket E \vdash c : \rho \rrbracket_f \triangleright \overline{\langle\!\langle E \rangle\!\rangle} \otimes \langle\!\langle \rho \rangle\!\rangle_f$ *where* $s = \mathsf{sec}(\rho)$.

---

[7] However we *can* add secrecy levels to constants, which may conform to the approaches in secure function calculi [2, 42].

Proof: The proof is by induction on the length of the derivation $E \vdash c : \rho$, uses the Eval lemma, and the results of the examples in section 4.4. When the last rule is (`subs`), we note that the original rule can be decomposed into two rules:

$$\texttt{(subs-1)}\ \frac{E \vdash c : s\ \texttt{cmd}\Downarrow \quad s' \leq s}{\vdash c : s'\ \texttt{cmd}\Downarrow} \qquad\qquad \texttt{(subs-2)}\ \frac{E \vdash c : s\ \texttt{cmd}\Downarrow \quad s' \leq s}{\vdash c : s\ \texttt{cmd}\Uparrow}$$

In both cases we use degradation. In second case we note that security levels are arbitrary for linear types, and that all other cases (including assign) work when f is linear as well as non-linear. See Appendix E.4.5 for details. ■

Note the sequent uses the dual of $\langle\!\langle E \rangle\!\rangle$, since $\langle\!\langle E \rangle\!\rangle$ itself represents the behaviour of the variables. The sequent makes explicit the decomposition of the original command type, first as the interface type, second as its secrecy index, and third as the action type.

In terms of the soundness of subsumption mentioned above, it is interesting to observe how the encoding illustrates the reason why the divergent command type cannot be elevated as the convergent command types. Let $\rho = s\ \texttt{cmd}\Uparrow$ in the translation of $E \vdash \texttt{while}\ e\ \texttt{do}\ c : \rho$ in Figure 4. Then we can see, in the encoding, the body of the loop, which is at level s, depends on the branching at level $\texttt{sec}(e)_E \leq s$: lowering s can make this dependency dangerous, hence we cannot degrade ρ as in the convergent types. Also note this argument does not use the restriction $s = \bot$ in the original type discipline.

Remark 6.5 The typability of the embedding suggests that the behaviours of the original typed terms would indeed be secure in terms of information flow. This can be formalised as a non-interference of the original system via the behavioural interference noted in Remark 5.9 and Theorem 6.4, together with a few basic equational and operational correspondences between original terms and their encoding (using the map of the environment in Figure 4). This is the possibilistic non-interference for the multi-threaded imperative calculus, first established in [50]:

$$E \vdash \sigma_1 \sim_s \sigma_2 \ \wedge\ E \vdash c : \rho \ \wedge\ (c, \sigma_1) \twoheadrightarrow \sigma_1' \quad \Rightarrow \quad (c, \sigma_2) \twoheadrightarrow \sigma_2' \text{ s.t. } E \vdash \sigma_1' \sim_s \sigma_2' \ .$$

Here $\sigma_1 \sim_s \sigma_2$ means σ_1 and σ_2 differ only in variables at levels above s. The result holds for all terms typable in rules in Figure 3, including typed terms not coming from [50]. We note the arguments in the proof do not depend on the restriction to boolean types; and that the same result holds for two different operational semantics for the original language.

6.3. Termination as Observables

After the preceding development, a natural question is whether we obtain any new information by doing such an endeavour or not. We think about this question in this subsection, and present a technical development which may answer affirmatively to this question.

We first return to the restriction of the original system where we only allow the level $\bot$ for divergent commands. This does seem a strong constraint, especially with multiple security levels. Now how does this constraint appear in process representation? It means we only assign $()^{\Uparrow}_{\bot}$ to a channel signalling the termination. Note this formulation makes explicit the notion of termination observables, both as types and as behaviours, which is somehow implicit in the system in [50]. Once we have this notion, we ask what is the real content of having the observable only at $\bot$. Clearly the answer is: "we allow *everybody* to observe the termination," from which the above noted restriction necessarily arises. We may then ask what would be the outcome of *not* allowing everybody to observe the termination. Can this make sense? It seems it does: since the time of Multics and as was recently introduced in a widely known program language [18], a mechanism by which we can prevent processes from even realising the presence of other processes, depending on assigned security levels, is a well-established idea in security, both from integrity and secrecy concerns. Thus we reason this extension is meaningful practically.

Further, there is a technically important observation, which is that the encoding in Figure 4 does *not* apparently impose restriction on levels of divergent types. Indeed the argument for Theorem 6.4 hardly depends on such conditions. Thus we may generalise the original `while` rule in Figure 3 as follows.

$$(\mathtt{while})\quad \frac{E \vdash e : s \qquad E \vdash c : s\ \mathtt{cmd}_{\Uparrow}}{E \vdash \mathtt{while}\ e\ \mathtt{do}\ c : s\ \mathtt{cmd}_{\Uparrow}}$$

The new rule is particularly significant in its loosened condition on the guard of the loop, allowing us to type the following term, where we write M for a secrecy level between H and L.

$$\mathtt{while}\ e^{\mathrm{M}}\ \mathtt{do}\ c\ :\ \mathrm{H}\ \mathtt{cmd}_{\Uparrow}$$

The program seems to give a reasonable behaviour, using low-level data for high-level purposes, and, as far as its termination cannot be seen by the observers below M (or type-wise below H), the secrecy is indeed preserved.[8] Based on the above rule we can naturally extend the system to allow multiple threads to own different levels of termination observables, which we do not discuss here. Without changing the encoding, we obtain the same soundness result for the extended system.

Theorem 6.6 (soundness in the extended system) *If* $E \vdash c : \rho$ *in the extended system, then* $[\![E]\!] \cdot f : [\![\rho]\!] \vdash_s [\![E \vdash c : \rho]\!]_f \triangleright \overline{\langle\!\langle E \rangle\!\rangle} \otimes \langle\!\langle \rho \rangle\!\rangle_f$ *where* $s = \mathsf{sec}(\rho)$.

PROOF: See Appendix E.4.5. ■

Significantly this result leads to, via the behavioural non-interference mentioned in Remark 5.9, a non-interference result for the extended imperative calculus. The formulation is different since the termination behaviours can change between two initial configurations if we set different values at levels lower than the termination observable. The property can be formulated as follows. Let $E \vdash c : \rho$ with $\mathsf{sec}(\rho) = s$ and $\sigma_1 \sim_{s'} \sigma_2$. Then there is a relation $(c, \sigma_1)\mathbf{R}(c, \sigma_2)$ s.t. whenever $(c_1, \sigma_1')\,\mathbf{R}\,(c_2, \sigma_2')$ we have:

(i) $E \vdash c_i : \rho'$ $(i = 1, 2)$ with $\mathsf{sec}(\rho') = s$, as well as $\sigma_1' \sim_{s'} \sigma_2'$.

(ii) If $(c_1, \sigma_1') \twoheadrightarrow (c_1', \sigma_1'')$ then $(c_2, \sigma_2') \twoheadrightarrow (c_2', \sigma_2'')$ such that $(c_1', \sigma_1'')\mathbf{R}(c_2', \sigma_2'')$, and the symmetric case.

(iii) Let $s \leq s'$. Then whenever $(c_1, \sigma_1') \twoheadrightarrow \sigma_1''$ we have $(c_2, \sigma_2') \twoheadrightarrow \sigma_2''$ such that $\sigma_1'' \sim_{s'} \sigma_2''$.

Since (i) and (ii) also hold in the original calculus, here we are having a weakened condition in the case of $s \not\leq s'$ where the property in (iii) may not hold. Note (iii) says that the termination observable is the same as far as the given two environments are equivalent for those who can observe the termination. Thus we are again guaranteed secure information flow with added typability, by starting from a typed process representation of imperative program behaviour. Articulating and controlling observables using the present framework opens other opportunities.

---

[8] We note that, while the resulting system looks somewhat similar to the sequential system in [50], there is a basic difference in that the present extended system does *not* allow a program like x^{L} :=`true`; `while` b^{H} `do skip end`; x^{L} :=`false`. This is because, in the present extended system, we cannot lower the level of the while command in the middle to L. More intuitively, this program is not secure since another low-level thread may observe what is in b if ever x turns false.

A. Untyped Terms

We first reproduce the syntax of untyped terms which are restricted to those which own no free output.

$$\begin{array}{lll} P & ::= & x(\vec{y}).P \mid \overline{x}\langle(\boldsymbol{\nu}\,\vec{z})\vec{y}\rangle.P \mid x[(\vec{y}).P \,\&\, (\vec{z}).Q] \mid \overline{x}\,\mathtt{inl}(\boldsymbol{\nu}\,\vec{z}).P \mid \overline{x}\,\mathtt{inr}(\boldsymbol{\nu}\,\vec{z}).P \mid \\ & & P \mid Q \mid (\boldsymbol{\nu}\,x)P \mid (\mu X(\vec{x}).P)\langle\vec{y}\rangle \mid X\langle\vec{x}\rangle \mid \mathbf{0} \end{array}$$

Now the *reduction relation* over untyped terms is the smallest relation closed under the following rules (Below we omit the obvious (Com$_r$) and (Rec$_r$)).

$$\begin{array}{lll} (\textsc{Com}) & x(\vec{y}).P \mid \overline{x}(\boldsymbol{\nu}\,\vec{y}).Q \longrightarrow (\boldsymbol{\nu}\,\vec{y})(P \mid Q) & \\ (\textsc{Com}_l) & x[(\vec{y}_1).P_1 \,\&\, (\vec{y}_2).P_2] \mid \overline{x}\mathtt{inl}(\boldsymbol{\nu}\,\vec{y}_1).Q \longrightarrow (\boldsymbol{\nu}\,\vec{y}_1)(P_1 \mid Q) & \\ (\textsc{Rec}) & \mathcal{E}\langle\vec{x}\rangle \mid \overline{x_0}(\boldsymbol{\nu}\,\vec{y}).Q \longrightarrow (\boldsymbol{\nu}\,\vec{y})(P\{\mathcal{E}/X\} \mid Q) & \mathcal{E} = \mu X(\vec{x}).x_0(\vec{y}).P \\ (\textsc{Rec}_l) & \mathcal{E}\langle\vec{x}\rangle \mid \overline{x_0}\mathtt{inl}(\boldsymbol{\nu}\,\vec{y}_1).Q \longrightarrow (\boldsymbol{\nu}\,\vec{y}_1)(P_1\{\mathcal{E}/X\} \mid Q) & \mathcal{E} = \mu X(\vec{x}).x_0[(\vec{y}_1).P_1 \,\&\, (\vec{y}_2).P_2] \end{array}$$

$$(\textsc{Par})\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad (\textsc{Res})\ \frac{P \longrightarrow P'}{(\boldsymbol{\nu}\,a)P \longrightarrow (\boldsymbol{\nu}\,a)P'} \qquad (\textsc{Str})\frac{P' \equiv P \quad P \longrightarrow Q \quad Q \equiv Q'}{P' \longrightarrow Q'}$$

where $\equiv$ is the smallest congruence called *structural congruence* closed under the following rules.

$(\equiv_\alpha)$ $P \equiv Q$ if $P \equiv_\alpha Q$.

(Par) $P \mid \mathbf{0} \equiv P$, $P \mid Q \equiv Q \mid P$ and $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$.

(Res) $(\boldsymbol{\nu}\,x, y)P \equiv (\boldsymbol{\nu}\,y, x)P$, $(\boldsymbol{\nu}\,x)\mathbf{0} \equiv \mathbf{0}$, and $(\boldsymbol{\nu}\,x)P \mid Q \equiv (\boldsymbol{\nu}\,x)(P \mid Q)$ if $x \notin \mathsf{fn}(Q)$.

B. Preterms

The syntax of preterms is given by:

$$\begin{array}{lll} P & ::= & x(\vec{y} : \vec{\tau}).P \mid \overline{x}(\boldsymbol{\nu}\,\vec{z} : \vec{\tau}).P \mid x[(\vec{y} : \vec{\tau}).P \,\&\, (\vec{z} : \vec{\tau}).Q] \mid \overline{x}\,\mathtt{inl}(\boldsymbol{\nu}\,\vec{z} : \vec{\tau}).P \mid \overline{x}\,\mathtt{inr}(\boldsymbol{\nu}\,\vec{z} : \vec{\tau}).P \\ & \mid & P \mid Q \mid (\boldsymbol{\nu}\,x)P \mid (\mu X(\vec{x}).P)\langle\vec{y}\rangle \mid X\langle\vec{x}\rangle \mid \mathbf{0} \end{array}$$

where we assume that, in each pair of vectors of form $\vec{y} : \vec{\rho}$, that they have the same length, and that names in each pair of parentheses are pairwise distinct. The structural congruence $\equiv$ on preterms is defined as in Appendix A, except: (1) The congruence is considered in terms of well-typed contexts. (2) We add: "$P \equiv Q$ iff they are identical modulo the equality on types generated by $\langle\tau, \tau'\rangle = \langle\tau', \tau\rangle$". (3) The rules (Res) are replaced by $(\boldsymbol{\nu}\,x : \alpha)(\boldsymbol{\nu}\,y : \beta)P \equiv (\boldsymbol{\nu}\,y : \beta)(x : \alpha)P$, $(\boldsymbol{\nu}\,x : \alpha)\mathbf{0} \equiv \mathbf{0}$, and $(\boldsymbol{\nu}\,x : \alpha)P|Q \equiv (\boldsymbol{\nu}\,x : \alpha)(P|Q)$ if $x \notin \mathsf{fn}(Q)$. We then define the reduction on preterms as in untyped terms except (Com) is replaced by:

$$(\textsc{Com}) \quad x(\vec{y} : \vec{\tau}_1).P \mid \overline{x}(\boldsymbol{\nu}\,\vec{y} : \vec{\tau}_2).Q \longrightarrow (\boldsymbol{\nu}\,\vec{y} : \langle\vec{\tau}_1,\ \vec{\tau}_2\rangle)(P \mid Q)$$

with $\langle\vec{\tau},\ \vec{\tau'}\rangle$ denoting pairs of respective types. Similarly for (Com$_{l,r}$), (Rec) and (Rec$_{l,r}$).

Reduction on preterms and that on untyped terms are closely related. To formalise this relationship, we define $\mathtt{strip}(P)$ as the result of stripping off all type annotations from P (thus obtaining the untyped counterpart of P). Then we note, letting $P, P', \ldots$ range over preterms and $U, U', \ldots$ over untyped terms,

(i) $P \equiv P'$ implies $\mathtt{strip}(P) \equiv \mathtt{strip}(P')$. Similarly $P \longrightarrow P'$ implies $\mathtt{strip}(P) \longrightarrow \mathtt{strip}(P')$.

(ii) Let $U = \mathtt{strip}(P)$. Then $U \equiv U'$ implies $P \equiv P'$ such that $\mathtt{strip}(P') = U'$. Similarly $U \longrightarrow U'$ implies $P \longrightarrow P'$ such that $\mathtt{strip}(P') = U'$.

By these properties we safely commute between preterms and untyped terms. As an example, let us write $\Gamma \vdash_s U \triangleright A$ for an untyped term U iff $\Gamma \vdash_s P \triangleright A$ with $U = \mathtt{strip}(P)$. Then the subject reduction in Section 5 immediately gives us the following:

$$\Gamma \vdash_s U \triangleright A \text{ and } U \longrightarrow U' \text{ with } \mathsf{bn}(U') \cap \mathsf{fn}(\Gamma) = \emptyset \text{ implies } \Gamma \vdash_s U' \triangleright A.$$

C. Typing System

(Common Rules)

(Zero)

$$\frac{}{\Gamma \vdash_s \mathbf{0} \triangleright \emptyset}$$

(Par) $A_1 \asymp A_2$

$$\frac{\Gamma \vdash_s P_i \triangleright A_i \quad (i=1,2)}{\Gamma \vdash_s P_1 | P_2 \triangleright A_1 \odot A_2}$$

(Res)

$$\frac{\Gamma \cdot x : \alpha \vdash_s P \triangleright A \otimes \mathsf{p}x \quad \mathsf{p} \in \{\updownarrow, \mathbf{!}, \Updownarrow\}}{\Gamma \vdash_s (\boldsymbol{\nu}\, x : \alpha) P \triangleright A}$$

(Deg$_s$)

$$\frac{\begin{array}{l}\Gamma \vdash_s P \triangleright A \\ s' \leq s\end{array}}{\Gamma \vdash_{s'} P \triangleright A}$$

(Weak-$\updownarrow$)

$$\frac{\begin{array}{l}\Gamma \vdash_s P \triangleright A^{-x} \\ \mathsf{md}(\Gamma(x)) = \{\downarrow, \uparrow\}\end{array}}{\Gamma \vdash_s P \triangleright A \otimes \updownarrow x}$$

(Weak-$\mathbf{?}$)

$$\frac{\begin{array}{l}\Gamma \vdash_s P \triangleright A \quad x \notin \mathsf{sbj}(A) \\ \mathbf{?} \in \mathsf{md}(\Gamma(x)) \quad \mathsf{p} \in \{\mathbf{?}, \mathbf{?}^{\iota}\}\end{array}}{\Gamma \vdash_s P \triangleright A \otimes \mathsf{p}x}$$

(Weak-$\Updownarrow$)

$$\frac{\begin{array}{l}\Gamma \vdash_s P \triangleright A \quad x \notin \mathsf{sbj}(A) \\ \Gamma(x) \text{ nonlinear}\end{array}}{\Gamma \vdash_s P \triangleright A \otimes \Updownarrow x}$$

(Non-Linear Rules)

(In)

$$\frac{\begin{array}{l}\vdash (\vec{\tau})^{\Downarrow}_{s} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \vdash_s P \triangleright \mathsf{p}\vec{y} \otimes \mathbf{?}A \otimes \Updownarrow x\end{array}}{\Gamma \vdash_s x(\vec{y} : \vec{\tau}).P \triangleright A \otimes \Updownarrow x}$$

(Out)

$$\frac{\begin{array}{l}\vdash (\vec{\tau})^{\Uparrow}_{s} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \vdash_s P \triangleright \mathsf{p}\vec{y} \otimes \mathbf{?}A \otimes \Updownarrow x\end{array}}{\Gamma \vdash_s \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\alpha}).P \triangleright A \otimes \Updownarrow x}$$

(Var)

$$\frac{\begin{array}{l}\vdash \alpha_i \leq \Gamma(x_i) \quad \mathsf{md}(\alpha_0) = \Downarrow \\ \begin{cases} \mathsf{p}_i = \Updownarrow & (\alpha_i \text{ nonlinear}) \\ \mathsf{p}_i = \mathsf{md}(\alpha_i) = \mathbf{?} & (\text{else})\end{cases}\end{array}}{\Gamma \cdot X : \vec{\alpha} \vdash_{\mathsf{sec}(\alpha_0)} X\langle \vec{x} \rangle \triangleright \mathsf{p}\vec{x}}$$

(Bra) $\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\Downarrow}_{s} \leq \Gamma(x)$

$$\frac{\Gamma \cdot \vec{y}_i : \vec{\tau}_i \vdash_s P_i \triangleright \mathsf{p}\vec{y}_i \otimes \mathbf{?}A \otimes \Updownarrow x}{\Gamma \vdash_s x[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2] \triangleright A \otimes \Updownarrow x}$$

(Sel$_l$) $\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\Uparrow}_{s} \leq \Gamma(x)$

$$\frac{\Gamma \cdot \vec{y}_1 : \vec{\tau}_1 \vdash_s P \triangleright \mathsf{p}\vec{y}_1 \otimes \mathbf{?}A \otimes \Updownarrow x}{\Gamma \vdash_s \overline{x}\texttt{inl}(\boldsymbol{\nu}\, \vec{y}_1 : \vec{\tau}_1).P \triangleright A \otimes \Updownarrow x}$$

(Rec) $\vdash \alpha_i \leq \Gamma(z_i)$ α_0 nonlinear

$$\frac{\Gamma\{\vec{x}/\vec{z}\} \cdot X : \vec{\alpha} \vdash_s P \triangleright \mathbf{?}A\{\vec{x}/\vec{z}\}}{\Gamma \vdash_s (\mu X(\vec{x} : \vec{\alpha}).P)\langle \vec{z} \rangle \triangleright A}$$

(Linear Rules) We omit `(Sel`$^{\uparrow}_{r}$`)` and `(Sel`$^{\mathbf{?}}_{r}$`)`.

(In$^{\downarrow}$) (where $C/\vec{y} = \downarrow\uparrow B$)

$$\frac{\begin{array}{l}\vdash (\vec{\tau})^{\downarrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \vdash_s P \triangleright \mathbf{?}A \otimes C^{\text{-}x}\end{array}}{\Gamma \vdash_s x(\vec{y} : \vec{\tau}).P \triangleright A \otimes \downarrow x \rightarrow B}$$

(Out$^{\uparrow}$) (where $C/\vec{y} = \downarrow\uparrow B$)

$$\frac{\begin{array}{l}\vdash (\vec{\tau})^{\uparrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \vdash_s P \triangleright \mathbf{?}A \otimes C^{\text{-}x}\end{array}}{\Gamma \vdash_s \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\tau}).P \triangleright A \otimes \uparrow x \rightarrow B}$$

(Var$^{!}$) $\vdash \alpha_i \leq \Gamma(x_i)$

$$\frac{\begin{array}{l}\mathsf{md}(\alpha_0) = \mathbf{!} \\ \mathsf{md}(\alpha_i) \in \{\mathbf{?}, \Downarrow, \Uparrow\} \ (i \neq 0)\end{array}}{\Gamma \cdot X : \vec{\alpha} \vdash_s X\langle \vec{x} \rangle \triangleright X\langle \vec{x} \rangle}$$

(Bra$^{\downarrow}$) (where $C_i/\vec{y}_i = \downarrow\uparrow B$)

$$\frac{\begin{array}{l}\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\downarrow}_{s} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}_i : \vec{\tau}_i \vdash_s P_i \triangleright \mathbf{?}A \otimes C_i^{\text{-}x} \quad (i=1,2)\end{array}}{\Gamma \vdash_s x[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2] \triangleright A \otimes \downarrow x \rightarrow B}$$

(Sel$^{\uparrow}_{l}$) (where $C/\vec{y}_1 = \downarrow\uparrow B$)

$$\frac{\begin{array}{l}\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\uparrow}_{s} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}_1 : \vec{\tau}_1 \vdash_s P \triangleright \mathbf{?}A \otimes C^{\text{-}x}\end{array}}{\Gamma \vdash_s \overline{x}\texttt{inl}(\boldsymbol{\nu}\, \vec{y}_1 : \vec{\tau}_1).P \triangleright A \otimes \uparrow x \rightarrow B}$$

(In$^{!}$)

$$\frac{\begin{array}{l}\vdash (\vec{\tau})^{\mathbf{!}}_{s,\kappa} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i) \\ \begin{cases} \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \vdash_s P \triangleright \mathsf{p}\vec{y} \otimes \mathbf{?}^{\iota}A\{\vec{x}/\vec{z}\} \otimes X\langle \vec{x} \rangle & (\kappa = \boldsymbol{\iota}) \\ \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \vdash_s P \triangleright \mathsf{p}\vec{y} \otimes \mathbf{?}A\{\vec{x}/\vec{z}\} \otimes X\langle x\vec{w} \rangle & (\kappa = \boldsymbol{\mu})\end{cases}\end{array}}{\Gamma \vdash_s (\mu X(\vec{x} : \vec{\alpha}).x_0(\vec{y} : \vec{\tau}).P)\langle \vec{z} \rangle \triangleright \mathbf{!}z_0 \otimes A}$$

(Out$^{?}$) (where $C/\vec{y} = \downarrow\uparrow B$)

$$\frac{\begin{array}{l}\vdash (\vec{\tau})^{\mathbf{?}}_{s',\kappa} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \vdash_s P \triangleright \mathbf{?}A \otimes C \otimes \mathtt{p}x \quad \mathtt{p} \in \{\mathbf{?}, \mathbf{?}^{\iota}\} \\ \kappa = \boldsymbol{\mu} \Rightarrow (s = s' \wedge \mathtt{p} = \mathbf{?})\end{array}}{\Gamma \vdash_s \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\tau}).P \triangleright A \otimes B \otimes \mathtt{p}x}$$

(Bra$^{!}$)

$$\frac{\begin{array}{l}\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\mathbf{!}}_{s,\kappa_1 \& \kappa_2} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i) \qquad (j=1,2) \\ \begin{cases} \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \vdash_s P_j \triangleright \mathsf{p}\vec{y}_j \otimes \mathbf{?}^{\iota}A_j\{\vec{x}/\vec{z}\} \otimes X\langle \vec{x} \rangle & (\kappa_j = \boldsymbol{\iota}) \\ \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \vdash_s P_j \triangleright \mathsf{p}\vec{y}_j \otimes \mathbf{?}A_j\{\vec{x}/\vec{z}\} \otimes X\langle x\vec{w}_j \rangle & (\kappa_j = \boldsymbol{\mu})\end{cases}\end{array}}{\Gamma \vdash_s (\mu X(\vec{x} : \vec{\alpha}).x_0[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2])\langle \vec{z} \rangle \triangleright \mathbf{!}z_0 \otimes (A_1 \odot A_2)}$$

(Sel$^{?}_{l}$) (where $C/\vec{y} = \downarrow\uparrow B$)

$$\frac{\begin{array}{l}\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\mathbf{?}}_{s',\kappa_1 \oplus \kappa_2} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}_1 : \vec{\tau}_1 \vdash_s P \triangleright \mathbf{?}A \otimes C \otimes \mathtt{p}x \quad \mathtt{p} \in \{\mathbf{?}, \mathbf{?}^{\iota}\} \\ \kappa_1 = \boldsymbol{\mu} \Rightarrow (s = s' \wedge \mathtt{p} = \mathbf{?})\end{array}}{\Gamma \vdash_s \overline{x}\texttt{inl}(\boldsymbol{\nu}\, \vec{y}_1 : \vec{\tau}_1).P \triangleright A \otimes B \otimes \mathtt{p}x}$$

D. The Minimum Typing System

(Common Rules)

(Zero)
$$\frac{}{\Gamma \Vdash_{\top} \mathbf{0} \triangleright \emptyset}$$

(Par)
$$\frac{\begin{array}{l} A_1 \asymp A_2 \\ \Gamma \Vdash_{s_i} P_i \triangleright A_i \quad (i=1,2) \end{array}}{\Gamma \Vdash_{s_1 \sqcap s_2} P_1 \mid P_2 \triangleright A_1 \odot A_2}$$

(Res) $\Gamma \cdot x : \alpha \Vdash_s P \triangleright A$
$$\frac{\left\{\begin{array}{l} A = \mathtt{p}x \otimes B,\ C = B,\ \mathtt{p} \in \{\updownarrow, !, \Updownarrow\} \\ A = C^{-x},\ \mathsf{md}(\alpha) = \{\uparrow, \downarrow\} \text{ or } \alpha \text{ nonlinear} \end{array}\right.}{\Gamma \Vdash_s (\boldsymbol{\nu}\, x : \alpha)P \triangleright C}$$

(Non-Linear Rules)

(In)
$$\frac{\begin{array}{l} \vdash (\vec{\tau})^{\Downarrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \Vdash_s P \triangleright ?A^{-x\vec{y}} \otimes B \\ B \leq_{\mathtt{G}} \overrightarrow{\mathtt{p}y} \otimes \Updownarrow x \quad s' \leq s \end{array}}{\Gamma \Vdash_{s'} x(\vec{y} : \vec{\tau}).P \triangleright A \otimes \Updownarrow x}$$

(Out)
$$\frac{\begin{array}{l} \vdash (\vec{\tau})^{\Uparrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \Vdash_s P \triangleright ?A^{-x\vec{y}} \otimes B \\ B \leq_{\mathtt{G}} \overrightarrow{\mathtt{p}y} \otimes \Updownarrow x \quad s' \leq s \end{array}}{\Gamma \Vdash_{s'} \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\tau}).P \triangleright A \otimes \Updownarrow x}$$

(Var) $\vdash \alpha_i \leq \Gamma(x_i)$
$$\frac{\begin{array}{l} \mathsf{md}(\alpha_0) = \Downarrow \\ \left\{\begin{array}{l} \mathtt{p}_i = \Updownarrow \quad (\alpha_i \text{ nonlinear}) \\ \mathtt{p}_i = \mathsf{md}(\alpha_i) = ? \text{ (else)} \end{array}\right. \end{array}}{\Gamma \cdot X : \vec{\alpha} \Vdash_{\mathsf{sec}(\alpha_0)} X\langle \vec{x} \rangle \triangleright \overrightarrow{\mathtt{p}x}}$$

(Bra) $\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\Downarrow}_{s'} \leq \Gamma(x)$
$$\frac{\begin{array}{l} \Gamma \cdot \vec{y}_i : \vec{\tau}_i \Vdash_{s_i} P_i \triangleright ?A_i^{-x\vec{y}_i} \otimes B \\ B_i \leq_{\mathtt{G}} \overrightarrow{\mathtt{p}y_i} \otimes \Updownarrow x \quad s' \leq s_1 \sqcap s_2 \quad A_1 \asymp A_2 \end{array}}{\Gamma \Vdash_{s'} x[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2] \triangleright A_1 \odot A_2 \otimes \Updownarrow x}$$

(Sel$_l$) $\vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\Uparrow}_{s'} \leq \Gamma(x)$
$$\frac{\begin{array}{l} \Gamma \cdot \vec{y}_1 : \vec{\tau}_1 \Vdash_s P \triangleright ?A^{-x\vec{y}_i} \otimes B \\ B \leq_{\mathtt{G}} \overrightarrow{\mathtt{p}y_1} \otimes \Updownarrow x \quad s' \leq s \end{array}}{\Gamma \Vdash_s \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, \vec{y}_1 : \vec{\tau}_1).P \triangleright A \otimes \Updownarrow x}$$

(Rec)
$$\frac{\begin{array}{l} \vdash \alpha_i \leq \Gamma(z_i) \qquad \alpha_0 \text{ nonlinear} \\ \Gamma\{\vec{x}/\vec{z}\} \cdot X : \vec{\alpha} \Vdash_s P \triangleright ?A\{\vec{x}/\vec{z}\} \end{array}}{\Gamma \Vdash_s (\mu X(\vec{x} : \vec{\alpha}).P)\langle \vec{z} \rangle \triangleright A}$$

(Linear Rules) We omit (`Sel`$^{\uparrow}_r$) and (`Sel`$^{?}_{l,r}$).

(In$^{\downarrow}$) $(C/\vec{y}' =_{\downarrow\uparrow} B,\ \{\vec{y}'\} \subseteq \{\vec{y}\}\ (\star))$
$$\frac{\begin{array}{l} \vdash (\vec{\tau})^{\downarrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \Vdash_s P \triangleright ?A \otimes C^{-x} \end{array}}{\Gamma \Vdash_s x(\vec{y} : \vec{\tau}).P \triangleright A \otimes \downarrow x \rightarrow B}$$

(Out$^{\uparrow}$) $(C/\vec{y}' =_{\downarrow\uparrow} B,\ \{\vec{y}'\} \subseteq \{\vec{y}\}\ (\star))$
$$\frac{\begin{array}{l} \vdash (\vec{\tau})^{\uparrow}_{s'} \leq \Gamma(x) \\ \Gamma \cdot \vec{y} : \vec{\tau} \Vdash_s P \triangleright ?A \otimes C^{-x} \end{array}}{\Gamma \Vdash_s \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\tau}).P \triangleright A \otimes \uparrow x \rightarrow B}$$

(Var$^{!}$) $\vdash \alpha_i \leq \Gamma(x_i)$
$$\frac{\begin{array}{l} \mathsf{md}(\alpha_0) = ! \\ \mathsf{md}(\alpha_i) \in \{?, \Downarrow, \Uparrow\}\ (i \neq 0) \end{array}}{\Gamma \cdot X : \vec{\alpha} \Vdash_{\top} X\langle \vec{x} \rangle \triangleright X\langle \vec{x} \rangle}$$

(Bra$^{\downarrow}$) $(C_i/\vec{y}_i' =_{\downarrow\uparrow} B,\ \{\vec{y}_i'\} \subseteq \{\vec{y}_i\}\ (\star))$
$$\frac{\begin{array}{l} \vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\downarrow}_{s_i} \leq \Gamma(x) \\ \Gamma \cdot \vec{y}_i : \vec{\tau}_i \Vdash_{s_i} P_i \triangleright ?A_i \otimes C_i^{-x}\ (i=1,2) \quad A_1 \asymp A_2 \end{array}}{\Gamma \Vdash_{s_1 \sqcap s_2} x[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2] \triangleright A_1 \odot A_2 \otimes \downarrow x \rightarrow B}$$

(Sel$^{\uparrow}_l$) $(C/\vec{y}_1' =_{\downarrow\uparrow} B,\ \{\vec{y}_i'\} \subseteq \{\vec{y}_i\}\ (\star))$
$$\frac{\begin{array}{l} \vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\uparrow}_{s} \leq \Gamma(x) \quad s = \mathsf{sec}(\Gamma(x)) \sqcap s_1 \\ \Gamma \cdot \vec{y}_1 : \vec{\tau}_1 \Vdash_{s_1} P \triangleright ?A \otimes C^{-x} \end{array}}{\Gamma \Vdash_s \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, \vec{y}_1 : \vec{\tau}_1).P \triangleright A \otimes \uparrow x \rightarrow B}$$

(In$^{!}$)
$$\frac{\begin{array}{l} \vdash (\vec{\tau})^{!}_{s,\kappa} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i) \quad B \leq_{\mathtt{G}} \overrightarrow{\mathtt{p}y} \\ \left\{\begin{array}{ll} \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \Vdash_s P \triangleright B \otimes ?^{\iota}A\{\vec{x}/\vec{z}\} \otimes X\langle \vec{x} \rangle & (\kappa = \boldsymbol{\iota}) \\ \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \Vdash_s P \triangleright B \otimes ?A\{\vec{x}/\vec{z}\} \otimes X\langle x_0\vec{w} \rangle & (\kappa = \boldsymbol{\mu}) \end{array}\right. \end{array}}{\Gamma \Vdash_s (\mu X(\vec{x} : \vec{\alpha}).x_0(\vec{y} : \vec{\tau}).P)\langle \vec{z} \rangle \triangleright !z_0 \otimes A}$$

(Out$^{?}$) $(C/\vec{y}' =_{\downarrow\uparrow} B,\ \{\vec{y}'\} \subseteq \{\vec{y}\}\ (\star))$
$$\frac{\begin{array}{l} \vdash (\vec{\tau})^{?}_{s_0,\kappa} \leq \Gamma(x) \quad \Gamma \cdot \vec{y} : \vec{\tau} \Vdash_s P \triangleright ?A^{-x} \otimes C' \\ \left\{\begin{array}{l} C' = C \otimes \mathtt{p}_0 x,\ \mathtt{p}_0 \in \{?, ?^{\iota}\} \\ C' = C^{-x} \end{array}\right. \\ \left\{\begin{array}{ll} \mathtt{p} = ?,\ s' = s \sqcap \mathsf{sec}(\Gamma(x)) & (\kappa = \boldsymbol{\mu}) \\ \mathtt{p}_1 = ?^{\iota}, \mathtt{p} = \mathtt{p}_0 \odot \mathtt{p}_1, s' = s & (\kappa = \boldsymbol{\iota}) \end{array}\right. \end{array}}{\Gamma \Vdash_{s'} \overline{x}(\boldsymbol{\nu}\, \vec{y} : \vec{\tau}).P \triangleright A \otimes B \otimes \mathtt{p}x}$$

(Bra$^{!}$) $\vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{!}_{s_j,\kappa_1 \& \kappa_2} \leq \Gamma(z_0) \quad \vdash \alpha_i \leq \Gamma(z_i)$
$$\frac{\begin{array}{l} B_j \leq_{\mathtt{G}} \overrightarrow{\mathtt{p}y_j} \quad (j=1,2) \quad A_1 \asymp A_2 \\ \left\{\begin{array}{ll} \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \Vdash_{s_j} P_j \triangleright B_j \otimes ?^{\iota}A_j\{\vec{x}/\vec{z}\} \otimes X\langle \vec{x} \rangle & (\kappa_j = \boldsymbol{\iota}) \\ \Gamma\{\vec{x}/\vec{z}\} \cdot \vec{y}_j : \vec{\tau}_j \cdot X : \vec{\alpha} \Vdash_{s_j} P_j \triangleright B_j \otimes ?A_j\{\vec{x}/\vec{z}\} \otimes X\langle x_0\vec{w}_j \rangle & (\kappa_j = \boldsymbol{\mu}) \end{array}\right. \end{array}}{\Gamma \Vdash_{s_1 \sqcap s_2} (\mu X(\vec{x} : \vec{\alpha}).x_0[(\vec{y}_1 : \vec{\tau}_1).P_1 \,\&\, (\vec{y}_2 : \vec{\tau}_2).P_2])\langle \vec{z} \rangle \triangleright !z_0 \otimes (A_1 \odot A_2)}$$

where $(\star)$ means $y_i \in \{\vec{y}'\}$ (or $y_{ij} \in \{\vec{y}_j{}'\}$) if $\mathsf{md}(\Gamma(y_i))$ (or $\mathsf{md}(\Gamma(y_{ij})) \in \{\downarrow, \uparrow, !\}$.

E. Proofs

E.1. Proofs for Section 3

E.1.1 Proof of Lemma 3.1

(1) By the induction on the derivation of $\leq$. The reflexive and anti-symmetry are obvious. Suppose $\vdash \tau \leq \tau'$ and $\vdash \tau' \leq \langle \tau_1,\ \tau_2 \rangle$. Then by (S-Single), we know $\vdash \tau' \leq \tau_1$ or $\vdash \tau' \leq \tau_2$. Then by inductive hypothesis, we have: $\vdash \tau \leq \tau_1$ or $\vdash \tau \leq \tau_2$. Hence by applying (S-Single) again, we have $\vdash \tau \leq \langle \tau_1,\ \tau_2 \rangle$, as desired. The case $\vdash \tau \leq \langle \tau_1,\ \tau_2 \rangle$ and $\vdash \langle \tau_1,\ \tau_2 \rangle \leq \langle \tau_1',\ \tau_2' \rangle$ is similar. Other cases are mechanical. (2) is mechanical by checking each rule. (3) is proved by the induction on the derivation of $\leq$. The interesting case is that types have linear modes. Assume $\vdash \alpha_{\mathtt{I}} \leq \alpha_{\mathtt{I}}'$ and $\vdash \alpha_{\mathtt{O}} \leq \alpha_{\mathtt{O}}'$ with $\vdash \alpha_{\mathtt{I}}' \asymp \alpha_{\mathtt{O}}'$. Then suppose $\alpha_{\mathtt{I}}' = (\vec{\tau}_1')^{\mathbf{!}}_{s_1'}$. Then by the subtype ordering defined in Figure 1, we can set $\alpha_{\mathtt{O}} = (\vec{\tau}_2')^{\mathbf{?}}_{s_2'}$ with $\vdash \tau_{1i}' \asymp \tau_{2i}'$ and $s_1' \geq s_2'$. Now by $\vdash \alpha_{\mathtt{I}} \leq \alpha_{\mathtt{I}}'$, we can write $\alpha_{\mathtt{I}} = (\vec{\tau}_1)^{\mathbf{!}}_{s_1}$ with $\vdash \tau_{1i} \leq \tau_{1i}'$ and $s_1 \geq s_1'$. Similarly we can set $\alpha_{\mathtt{O}} = (\vec{\tau}_2)^{\mathbf{!}}_{s_2}$ with $\vdash \tau_{2i} \leq \tau_{2i}'$ and $s_2 \leq s_2'$. By inductive hypothesis, we have $\vdash \tau_{1i} \asymp \tau_{2i}$. Also we have $s_1 \geq s_1' \geq s_2' \geq s_2$. Hence we have $\vdash \alpha_{\mathtt{I}} \asymp \alpha_{\mathtt{O}}$, as desired. The case $m_{\mathtt{I}} = \downarrow$ is just similar. (4) By induction of the derivation of $\leq$. Only interesting case is $\alpha = \langle \alpha_{\mathtt{I}},\ \alpha_{\mathtt{O}} \rangle$, which can be proved by (3).

E.1.2 Proof of Lemma 3.4

In this and the following proofs, we first note that, by definition, a well-formed action graph A takes the form of, with all subjects distinct,

$$A \quad = \quad \underbrace{\overrightarrow{\Updownarrow x} \otimes \overrightarrow{\mathbf{?}z} \otimes \underbrace{\overrightarrow{\mathbf{?}^{\iota} v}}_{\mathbf{?}^{\iota} B}}_{\mathbf{?}C} \otimes \overrightarrow{\mathbf{!}w} \otimes \overrightarrow{\updownarrow y} \otimes (X\langle \vec{u} \rangle \otimes) \downarrow\uparrow E$$

where $(X\langle \vec{u} \rangle \otimes)$ shows the item may or may not exist.

Moving to the proof of Lemma 3.4, (1) and (2) are straightforward. For (3), first it is obvious that if $\mathsf{sbj}(A_1) \cap \mathsf{sbj}(A_2) = \emptyset$ and $\mathsf{fv}(A_1) \cap \mathsf{fv}(A_2) = \emptyset$, then we always have $A_1 \asymp A_2$ and $A_1 \odot A_2 = A_1 \otimes A_2$ is well-formed. So suppose $\mathtt{p}x \in A_1$ and $\mathtt{q}x \in A_2$. We shall prove this case by the induction on the number of nodes of action graphs A_1 and A_2, noting $\mathtt{number}(|A_1|) + \mathtt{number}(|A_2|) \geq \mathtt{number}(|A_1 \odot A_2|)$. By definition we have the following combinations for $\mathtt{p}$ and $\mathtt{q}$.

1. $\mathtt{p} = \mathbf{!}$, $\mathtt{q} = \mathbf{?}$ 2. $\mathtt{p} = \mathbf{!}$, $\mathtt{q} = \mathbf{?}^{\iota}$ 3. $\mathtt{p} = \mathbf{?}$, $\mathtt{q} = \mathbf{?}$ 4. $\mathtt{p} = \mathbf{?}$, $\mathtt{q} = \mathbf{?}^{\iota}$
5. $\mathtt{p} = \mathbf{?}^{\iota}$, $\mathtt{q} = \mathbf{?}^{\iota}$ 6. $\mathtt{p} = \Updownarrow$, $\mathtt{q} = \Updownarrow$ 7. $\mathtt{p} = \downarrow$, $\mathtt{q} = \uparrow$

and the symmetric cases of 1, 2, 4 and 7. Suppose the case 1. Then we can write $A_1 = \mathbf{!}x \otimes A_1'$ and $A_2 = \mathbf{?}x \otimes A_2'$. Since $A_1 \asymp A_2$ implies $A_1' \asymp A_2'$, $A_1' \odot A_2'$ is well-formed by inductive hypothesis. Now by $\mathbf{!} \odot \mathbf{?} = \mathbf{!}$, we have $A_1 \odot A_2 = \mathbf{!}x \otimes A_1' \odot A_2'$. Since $x \notin \mathsf{sbj}(A_1') \cup \mathsf{sbj}(A_2')$, $\mathbf{!}x \otimes A_1' \odot A_2'$ is well-formed, and $\mathtt{number}(|A_1 \odot A_2|) = 1 + \mathtt{number}(|A_1' \odot A_2'|) \leq 1 + \mathtt{number}(|A_1'|) + \mathtt{number}(|A_2'|) \leq \mathtt{number}(|A_1|) + \mathtt{number}(|A_2|)$. Cases 2–6 are the same.

Now suppose the case 7. The only interesting case is that a new edge is created after the composition (if not, it is the same as the above). Here we only have to note that a cyclic relation is not newly created. Then the only possible case is that there exist $\mathbf{n}_1, \mathbf{n}_2$ such that $\mathbf{n}_1 \leq_{A_1} \mathtt{p}x$ and $\mathtt{q}x \leq_{A_2} \mathbf{n}_2$ with $\mathtt{p} \asymp \mathtt{q}$, $\mathbf{n}_1 \leq_{A_1 \odot A_2} \mathbf{n}_2$ and $\mathbf{n}_1, \mathbf{n}_2 \in |A_1 \odot A_2|$ (the case $\mathbf{n}_2 \leq_{A_2} \mathtt{p}x$ and $\mathtt{q}x \leq_{A_1} \mathbf{n}_1$ is just same). Then by well-formdness of A_1 and A_2, we know if $\mathbf{n}_1 =_{A_1} \mathtt{p}x$, then $\mathbf{n}_1 \notin |A_1 \odot A_2|$. Hence $\mathbf{n}_1 \lneq_{A_1} \mathtt{p}x$. Similarly we have $\mathtt{q}x \lneq_{A_2} \mathbf{n}_2$. Therefore by definition, $\mathbf{n}_1 \lneq_{A_1 \odot A_2} \mathbf{n}_2$ and $\mathsf{fn}(\mathbf{n}_1) \cap \mathsf{fn}(\mathbf{n}_2) = \emptyset$. Now we prove there is no edge such that $\mathbf{n}_2 \leq_{A_1 \odot A_2} \mathbf{n}_1$. If such an edge is created, then there are $\mathtt{p}'z$ and $\mathtt{q}'z$ such that $\mathbf{n}_2 \leq_{A_2} \mathtt{p}'z$ and $\mathtt{q}'z \leq_{A_1} \mathbf{n}_1$ with $\mathtt{p}' \asymp \mathtt{q}'$. Then by the transitivity of $\leq_{A_1}$ and $\leq_{A_2}$, originally there exist $\mathtt{q}'z \leq_{A_1} \mathtt{p}x$ and $\mathtt{q}x \leq_{A_2} \mathtt{p}'z$, which contradicts the condition (3) of $\asymp$. $\mathtt{number}(|A_1|) + \mathtt{number}(|A_2|) \geq \mathtt{number}(|A_1 \odot A_2|)$ is similarly proved.

E.1.3 Proof of Lemma 3.5

(i) is obvious by the symmetry of $\mathsf{p} \asymp \mathsf{q}$ and $\mathsf{p} \odot \mathsf{q}$. For (ii), first we note that the assumption immediately implies that $\mathsf{fv}(A_1) \cap \mathsf{fv}(A_2) \cap \mathsf{fv}(A_3) = \emptyset$ and $B_1 \cap B_2 \cap B_3 = \emptyset$ where $B_i = \{y \mid \downarrow y \in A_i \text{ or } \uparrow y \in A_i\}$ with $i = 1, 2, 3$. Hence if, for example, there exist $\mathbf{n}_1, \mathbf{n}_2$ such that $\mathbf{n}_1 \lneq_{A_1} \downarrow x$ and $\uparrow x \lneq_{A_2} \mathbf{n}_2$ with $\mathbf{n}_1 \lneq_{A_1 \odot A_2} \mathbf{n}_2$ and $\mathbf{n}_1, \mathbf{n}_2 \in |A_1 \odot A_2|$, it is not possible to have $\mathbf{n}_3$ such that $\mathbf{n}_3 \lneq_{A_3} \mathsf{p}x$ or $\mathsf{p}x \lneq_{A_3} \mathbf{n}_3$ for any p. Therefore (1) is obvious. For (2), by the similar reasoning of the proof of Lemma 3.5, we only have to check the acyclicity (the condition (3) of $\asymp$). Suppose as a contradiction, there are edges such that $\mathsf{p}_1 x \lneq_{A_1} \mathsf{q}_1 y$ and $\mathsf{p}y \lneq_{A_2 \odot A_3} \mathsf{q}x$. The case $\mathsf{p}y, \mathsf{q}x \in A_2$ obviously contradicts the assumption $A_1 \asymp A_2$. Similarly the case $\mathsf{p}y, \mathsf{q}x \in A_3$ contradicts (1). So assume $\mathsf{p}y \in A_2$ and $\mathsf{q}x \in A_3$. Then there exist $\mathsf{p}y \lneq_{A_2} \mathsf{p}'z$ and $\mathsf{q}'z \lneq_{A_3} \mathsf{q}x$ such that $\mathsf{p}' \asymp \mathsf{q}'$. Then noting $z \notin \mathsf{sbj}(A_1)$, $A_1 \asymp A_2$ implies $\mathsf{p}_1 x \lneq_{A_1 \odot A_2} \mathsf{p}'z$. This contradicts the assumption $A_1 \odot A_2 \asymp A_3$. (3) is easy by the uniqueness of $A \odot B$.

E.2. Proofs for Section 4

We show typability of agents given in Section 4.4 and used in Section 6. For the process and type abbreviations, refer to figure 2. In the proofs below we only show the layout of the deduction trees; the reader is invited to fill in the sequents.

E.2.1 Branching

Lemma E.1 (If) *If* $\Gamma \cdot b : \mathtt{bool}^{?}_{s'} \vdash_s P_i \triangleright \mathbf{?}A \otimes \downarrow\!\uparrow B \otimes \mathbf{?}^{\iota} b$ *for* $s' \leq s$, *then* $\Gamma \cdot b : \mathtt{bool}^{?}_{s'} \vdash_s \mathbf{If}\langle b, P_1, P_2 \rangle \triangleright A \otimes B \otimes \mathbf{?}^{\iota} b$.

Proof: The derivation tree is of the form

$$\dfrac{\begin{array}{c}\text{Hypothesis}\\ \text{Lemma 5.3 (weakening)}\end{array} \quad \begin{array}{c}\text{Hypothesis}\\ \text{Lemma 5.3 (weakening)}\end{array}}{\begin{array}{c}\text{Bra}^{\downarrow}\\ \text{Out}^{?}\end{array}}$$

Weakening places $c : \mathtt{bool}^{\downarrow}_{s'}$ in the basis, as required by the $\text{Bra}^{\downarrow}$-rule. The condition $s' \leq s$ comes into play at rule $\text{Bra}^{\downarrow}$ to show that $\mathtt{bool}^{!}_{s} \leq \mathtt{bool}^{!}_{s'}$. The innocuousness of b is used in rule $\text{Out}^{?}$ to show that $(\mathtt{bool}^{!}_{s'})^{?}_{\top} \leq (\mathtt{bool}^{!}_{s'})^{?}_{s'}$. For b non-innocuous we would not be able to show that $(\mathtt{bool}^{!}_{s'})^{?}_{s} \leq (\mathtt{bool}^{!}_{s'})^{?}_{s'}$.

E.2.2 Copy-cat

Let

$$[b^s \leftarrow b'^{s'}]^s \stackrel{\text{def}}{=} (\mu X(bb').b(c : \mathtt{bool}^{\uparrow}_{s}).(\mathbf{If}\langle b', \overline{c}\,\mathtt{inl}, \overline{c}\,\mathtt{inr}\rangle \mid X\langle bb'\rangle))\langle bb'\rangle$$

Lemma E.2 (Copy-cat) *If* $s' \leq s$, *then* $b : \mathtt{bool}^{!}_{s}, y : \mathtt{bool}^{?}_{s'} \vdash_s [b \leftarrow y] \triangleright \mathbf{!}b \otimes \mathbf{?}^{\iota} y$.

The derivation tree is of the form

$$\dfrac{\dfrac{\begin{array}{c}\text{Zero}\\ \text{Sel}^{\uparrow}_{l}\\ \text{Weak-}\mathbf{?}\end{array} \quad \begin{array}{c}\text{Zero}\\ \text{Sel}^{\uparrow}_{r}\\ \text{Weak-}\mathbf{?}\end{array}}{\text{Proposition E.1 (If)}} \qquad \text{Var}^{!}}{\begin{array}{c}\text{Par}\\ \text{In}^{\downarrow}\end{array}}$$

Start at each leaf with a basis of the form $X : \mathtt{bool}^{!}_{s}\mathtt{bool}^{?}_{s'} \cdot b : \mathtt{bool}^{!}_{s} \cdot y : \mathtt{bool}^{?}_{s'} \cdot c : \mathtt{bool}^{\uparrow}_{s}$. Rules Weak-**?** place $\mathbf{?}^{\iota} y$ in the action type, as required by the If-Lemma. The condition $s' \leq s$ is used in the If-Lemma. The subtyping pre-conditions at rules $\text{Sel}^{\uparrow}_{l}$, $\text{Sel}^{\uparrow}_{r}$, and $\text{In}^{\downarrow}$ are all trivial.

E.2.3 Imperative variable

Let

$$\mathbf{Var}\langle x^s b^{s'}\rangle^s \overset{\text{def}}{=} (\mu X(xb).x[(z:(\mathtt{bool}^{\mathbf{!}}_s)^{\uparrow}_s).(\overline{z}(\boldsymbol{\nu}\, b':\mathtt{bool}^{\mathbf{?}}_s).[b' \leftarrow b]|X\langle xb'\rangle)\ \&\ (b':\mathtt{bool}^{\mathbf{?}}_s).X\langle xb'\rangle])\langle xb\rangle$$

Lemma E.3 (Imperative variable) *If $s' \leq s$, then $x:\mathtt{var}^{\mathbf{!}}_s \cdot b:\mathtt{bool}^{\mathbf{!}}_{s'} \vdash_s \mathbf{Var}\langle xb\rangle \triangleright \mathbf{!}x \otimes \mathbf{?}^{\iota}b$.*

The derivation tree is of the form

$$\dfrac{\begin{array}{c}\text{Var}^{\mathbf{!}}\\ \text{Weak-}\mathbf{?}\end{array}\qquad \dfrac{\text{Var}^{\mathbf{!}}\qquad \begin{array}{c}\text{Proposition E.2.2 (Copy-cat)}\\ \text{Lemma 5.3 (weakening)}\\ \text{Out}^{\uparrow}\end{array}}{\text{Par}}}{\text{Bra}^{\mathbf{!}}}$$

Let Γ be the basis $X : \mathtt{var}^{\mathbf{!}}_s \cdot x : \mathtt{var}^{\mathbf{!}}_s \cdot b : \mathtt{bool}^{\mathbf{?}}_s$. Start at the left Var$^{\mathbf{!}}$ leaf with a basis $\Gamma \cdot b' : \mathtt{bool}^{\mathbf{?}}_{s'}$; start at the right Var$^{\mathbf{!}}$ leaf with a basis $\Gamma \cdot z : (\mathtt{bool}^{\mathbf{!}}_s)^{\uparrow}_{\top,\iota}$ (security levels are irrelevant for linear channels). Weakening rules underneath the Copy-cat yield a sequent with a basis Γ as required by the Par-rule. Rule Weak-$\mathbf{?}$ introduces $\mathbf{?}^{\iota}b$ in the action type, as required by the Bra$^{\mathbf{!}}$ rule. The condition $s' \leq s$ comes into play at the left Var$^{\mathbf{!}}$ leaf and at the Bra$^{\mathbf{!}}$ rule.

E.2.4 Reading a variable

Lemma E.4 (Read) *If $\Gamma \cdot x:\mathtt{var}_{s'} \cdot b:\mathtt{bool}^{\mathbf{?}}_{s''} \vdash_s P \triangleright \mathbf{?}A \otimes \langle\!\langle \tau \rangle\!\rangle_f \otimes \mathbf{?}^{\iota}b$ where $\mathbf{?}x \in A$, then $\Gamma \cdot x:\mathtt{var}_{s'} \vdash_s \mathbf{Read}\langle x, b, P\rangle \triangleright A \otimes \langle\!\langle \tau \rangle\!\rangle_f$.*

The derivation tree is of the form

$$\begin{array}{c}\text{Hypothesis}\\ \text{Lemma 5.3 (weakening)}\\ \text{In}^{\downarrow}\\ \text{Sel}^{\mathbf{?}}_{l}\end{array}$$

Weakening places $z:(\mathtt{bool}^{\mathbf{?}}_{s'})^{\downarrow}_{s'}$ in the basis.

E.2.5 Writing a variable

Lemma E.5 (Write) *If $\Gamma \cdot x:\mathtt{var}_s \cdot b:\mathtt{bool}^{\mathbf{?}}_{s'} \vdash_s P \triangleright \mathbf{?}A \otimes \langle\!\langle \tau \rangle\!\rangle_f \otimes \mathbf{?}^{\iota}b$ where $\mathbf{?}x \in A$, then $\Gamma \cdot x:\mathtt{var}_s \vdash_s \mathbf{Write}\langle b, x, P\rangle \triangleright A \otimes \langle\!\langle \tau \rangle\!\rangle_f \otimes \mathbf{?}^{\iota}b$.*

The derivation tree is of the form

$$\begin{array}{c}\dfrac{\begin{array}{c}\text{Hypothesis}\\ \text{Lemma 5.3 (weakening)}\end{array}\qquad \begin{array}{c}\text{Proposition E.2.2 (Copy-cat)}\\ \text{Lemma 5.3 (weakening)}\end{array}}{\text{Par}}\\ \text{Sel}^{\mathbf{?}}_{r}\end{array}$$

Weakening yields equal basis for the two branches, as required by the Par rule.

E.2.6 Assign

Lemma E.6 (Assign) *If $s' \leq s$, then $x:\mathtt{var}^{\mathbf{?}}_s \cdot y:\mathtt{var}^{\mathbf{?}}_{s'} \vdash_s \mathbf{Assign}\langle x^{s'} y^s\rangle \triangleright \mathbf{?}x \otimes \mathbf{?}^{\iota}y$.*

Use Proposition E.4 (Write) followed by Proposition E.4 (Read).

E.3. Proofs for Section 5

E.3.1 Proof of Lemma 5.3

For (1), first it is immediate that rules are closed under literal renaming (including bound names). Noting only those preterms who obey the binding convention are considered, the rest is mechanical by induction. (2) In the case (Res) and Weakenings, we use Lemma 3.1 (2) as well as the well-formedness of the action type A (Lemma 4.1 (1)). The other cases are a direct consequence of a condition of form $\vdash \alpha \leq \Gamma(x)$ or $\vdash \alpha \leq \Gamma(z_0)$ in the antecedent of each non-linear and linear rule. Similarly (3) is from the conditions on agent variable types. (4) is easy by induction. For (5), we note if $\Gamma \cdot x : \alpha \vdash_s P \triangleright A$ and $x \notin \mathsf{fn}(P)$, then we can always write $A = A' \otimes \mathsf{p}x$ with $\mathsf{p} \in \{\Updownarrow, \mathbf{?}, \mathbf{?}^\iota, \updownarrow\}$. Hence A/x is always defined and well-formed by Lemma 3.4 (2).

E.3.2 Proof of Lemma 5.4

(1) We prove the lemma by induction of the derivation of $\Gamma \vdash_s P \triangleright A$ (note that by definition of the syntax, P is an input guarded. However, in the non-linear case, this restriction is not important). By the definition of (Rec), the last rule should be either (Common Rules) except (Weak-$\updownarrow$), (Non-Linear Rules), (Out$^{\mathbf{?}}$) or (Sel$^{\mathbf{?}}$). The only interesting case is the last rule is (Var). Others are easy by inductive hypothesis. Assume the last derivation is (Var). Then we have: $\Gamma \cdot X : \vec{\alpha} \vdash_s X\langle\vec{w}\rangle \triangleright \overrightarrow{\mathsf{p}w}$ with $\mathsf{p}_0 = \Updownarrow$ and $\mathsf{p}_i = \mathbf{?}$ or $\Updownarrow$ $(i \geq 1)$. By assumption, we have: $\Gamma \vdash_s (\mu X(\vec{x}).X\langle\vec{w}\rangle)\langle\vec{x}\rangle \triangleright \overrightarrow{\mathsf{p}w}$. Now we prove $P\{\mathcal{E}/X\} = (\mu X(\vec{x}).X\langle\vec{w}\rangle)\langle\vec{w}\rangle$ has the action type A under Γ. By Lemma 5.3 (1), we have: $\Gamma\{\vec{w}/\vec{x}\} \vdash_s (\mu X(\vec{x}).X\langle\vec{w}\rangle)\langle\vec{w}\rangle \triangleright \mathsf{p}_0 w_0\{\vec{w}/\vec{x}\} \odot \mathsf{p}_1 w_1\{\vec{w}/\vec{x}\} \odot \cdots \odot \mathsf{p}_n w_n\{\vec{w}/\vec{x}\}$. Note that by the definition of $\mathcal{E}\langle\vec{w}\rangle$, $w_i \neq w_j$ iff $i \neq j$. Hence if $w_i = x_j$ for some i and j, then $i = j$. Therefore $\{\vec{w}/\vec{x}\}$ is the identity substitution, and $A\{\vec{w}/\vec{x}\} = A$. Hence we have $\Gamma\{\vec{w}/\vec{x}\} \vdash_s (\mu X(\vec{x}).X\langle\vec{w}\rangle)\langle\vec{w}\rangle \triangleright A$ as desired.

(2) Assume $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright \mathbf{?}A \otimes \mathbf{!}x_0$ with $\mathcal{E} \stackrel{\text{def}}{=} \mu X(\vec{x}).x_0(\vec{y}\!:\!\tau).P$ and $\vdash (\vec{\tau})^{!;s}_{\mu} \leq \Gamma(x_0)$. Then by the definition of the syntax and the form of the action type, we can set $P \stackrel{\text{def}}{=} X\langle x_0\vec{w}\rangle \,|\, Q$ with $\{\vec{w}\} \subseteq \{\vec{x} \cdot \vec{y}\}$, $\mathsf{fn}(Q) \subseteq \{\vec{x} \cdot \vec{y}\}$, and $\mathsf{fv}(Q) = \emptyset$. Hence we only have to show $\Gamma \cdot \vec{y}\!:\!\vec{\tau} \cdot X\!:\!\vec{\alpha} \vdash_s X\langle x_0\vec{w}\rangle \,|\, Q \triangleright \overrightarrow{\mathsf{p}y} \otimes \mathbf{?}A \otimes X\langle x_0\vec{w}\rangle$ implies $\Gamma \cdot \vec{y}\!:\!\vec{\tau} \vdash_s \mathcal{E}\langle x_0\vec{w}\rangle \,|\, Q \triangleright \overrightarrow{\mathsf{p}'y} \otimes \mathbf{?}A \otimes \mathbf{!}x_0$ with $\mathsf{p}'_i = \mathsf{p}_i$ or $\mathsf{p}'_i = \mathbf{?}$ if $\mathsf{p}_i = \mathbf{?}^\iota$. Since the cases that the last rule is either (Deg), (Weak-$\Updownarrow$) or (Weak-$\mathbf{?}$) is trivial, we can neglect these rules. Then by the assumption, there are derivations such that $\Gamma \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \vdash_s X\langle x_0\vec{w}\rangle \triangleright X\langle x_0\vec{w}\rangle$ and $\Gamma \cdot \vec{y} : \vec{\tau} \cdot X : \vec{\alpha} \vdash_s Q \triangleright \overrightarrow{\mathsf{p}y} \otimes \mathbf{?}A$ with $A \stackrel{\text{def}}{=} \overrightarrow{\mathsf{q}x}$ with $\mathsf{q}_i \in \{\mathbf{?}, \mathbf{?}^\iota, \Updownarrow\}$. Then by Lemma 5.3 (1), the assumption $\Gamma \vdash_s \mathcal{E}\langle\vec{x}\rangle \triangleright A \otimes \mathbf{!}x_0$ implies $\Gamma \vdash_s \mathcal{E}\langle x_0\vec{w}\rangle \triangleright \overrightarrow{\mathsf{q}w} \otimes \mathbf{!}x_0$. Now we prove that $\mathcal{E}\langle x_0\vec{w}\rangle \,|\, Q$ has the action type $\overrightarrow{\mathsf{p}'y} \otimes \overrightarrow{\mathsf{q}w} \otimes \mathbf{!}x_0$. Here we have to prove $(\overrightarrow{\mathsf{q}w} \otimes \mathbf{!}x_0) \odot (\overrightarrow{\mathsf{p}y} \otimes \overrightarrow{\mathsf{q}x}) = \overrightarrow{\mathsf{p}'y} \otimes \overrightarrow{\mathsf{q}w} \otimes \mathbf{!}x_0$. By the similar reasoning of the proof of the above (1), if $w_i = x_j$, then $i = j$ and $i \neq 0$, hence we have: $\mathsf{q}_i x_i \odot \mathsf{q}_j w_j$ is either (a) $\mathsf{q}_i x_i \otimes \mathsf{q}_j w_j$ with $x_i \neq w_j$ or (b) $\mathsf{q}_i x_i$ with $x_i = w_j$. In the case of (a), there must be some y_k such that $w_j = y_k$ since $\{\vec{w}\} \subseteq \{\vec{x} \cdot \vec{y}\}$. Hence $\mathsf{p}_k y_k \odot \mathsf{q}_j w_j = \mathsf{p}'_k y_k$ where we set $\mathsf{p}'_k = \mathbf{?}$ if $\mathsf{q}_j = \mathbf{?}$, else $\mathsf{p}'_k = \mathsf{p}_k$. Thus we are done.

(3) Similar to the proof of (2).

(4) This is a special case of (3) where $x_i = w_i$ for all i; hence $(\overrightarrow{\mathsf{q}x} \otimes \mathbf{!}x_0) \odot (\overrightarrow{\mathsf{p}y} \otimes \overrightarrow{\mathsf{q}x}) = \overrightarrow{\mathsf{p}y} \otimes \overrightarrow{\mathsf{q}x} \otimes \mathbf{!}x_0$, noting $\mathbf{?}A \odot \mathbf{?}A = \mathbf{?}A$.

(5) Similar to the proof of the case (4).

E.3.3 Proof of Lemma 5.5

By rule induction, referring to the structural rules in Appendix B. The commutativity $P \mid Q \equiv Q \mid P$ is by Lemma 3.5 (1), while the associativity $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ is by (2). For the rule of the scope opening, to prove that $\Gamma \vdash_s ((\boldsymbol{\nu}\, x : \alpha)P) \mid Q \triangleright A$ implies $\Gamma \vdash_s (\boldsymbol{\nu}\, x : \alpha)(P \mid Q) \triangleright A$ with $x \notin \mathsf{fn}(Q)$, we use (weakening), Lemma 5.3 (4). While the other direction is proved by (strengthening), Lemma 5.3 (5).

E.3.4 Proof of Subject Reduction Theorem (Theorem 5.6)

We consider only the monadic case. It is straightforward to extend the proof to the polyadic case. There are the following cases.

Non-linear Case: (1) (In)-(Out): Assume that

$$x(y\!:\!\tau_1).P_1 \mid \overline{x}(\boldsymbol{\nu}\, y\!:\!\tau_2).P_2 \longrightarrow (\boldsymbol{\nu}\, y\!:\!\langle\tau_1,\ \tau_2\rangle)(P_1 \mid P_2)$$

under the typing $\Gamma \Vdash_{s_1 \sqcap s_2} x(y : \tau_1).P_1 \mid \overline{x}(\boldsymbol{\nu}\, y : \tau_2).P_2 \triangleright A_1 \odot A_2 \otimes \Updownarrow x$ with the following derivations:

$$\begin{array}{lll} \Gamma \cdot y\!:\!\tau_1 \Vdash_{s_1'} P_1 \triangleright ?A_1 \otimes B_1 & \text{with} & \vdash (\tau_1)^{\Downarrow}_{s_1} \leq \Gamma(x) \\ \Gamma \cdot y\!:\!\tau_2 \Vdash_{s_2'} P_2 \triangleright ?A_2 \otimes B_2 & \text{with} & \vdash (\tau_2)^{\Uparrow}_{s_2} \leq \Gamma(x) \end{array}$$

where $B_i \leq_{\mathtt{G}} \mathtt{p}_i y \otimes \Updownarrow x$, $s_i \leq s_i'$ and $A_1 \asymp A_2$. By the subtyping rules of types in Figure 1 and Lemma 3.1 (3), we can derive $s_1 = s_2 = s_1 \sqcap s_2$ and $\tau_1 \asymp \tau_2$ from $\vdash (\tau_1)^{\Downarrow}_{s_1} \leq \Gamma(x)$ and $\vdash (\tau_1)^{\Uparrow}_{s_2} \leq \Gamma(x)$. By applying (subsumption) of Lemma 5.3 to the above derivations, we have:

$$\Gamma \cdot y\!:\!\langle\tau_1,\ \tau_2\rangle \Vdash_{s_1'} P_1 \triangleright ?A_1 \otimes B_1 \quad \text{and} \quad \Gamma \cdot y\!:\!\langle\tau_1,\ \tau_2\rangle \Vdash_{s_2'} P_2 \triangleright ?A_2 \otimes B_2$$

By Proposition 4.1 (1), we have the following cases for B_1 and B_2:

(a) τ_1 and τ_2 are nonlinear.

(b) $\mathsf{md}(\tau_i) = !$, $\mathsf{md}(\tau_j) = ?$ with $\mathtt{p}_i y \in B_i$ $(i \neq j,\ i, j \in \{1, 2\})$.

(c) $\mathsf{md}(\tau_i) = \uparrow$, $\mathsf{md}(\tau_j) \in \downarrow$ with $\mathtt{p}_i y \in B_i$ and $\mathtt{p}_j y \in B_j$ $(i \neq j,\ i, j \in \{1, 2\})$.

In each case, we can check $B_1 \asymp B_2$, and if $\mathtt{p}y \in B_1 \odot B_2$, then $\mathtt{p}$ is $\Updownarrow$ in (a), **!** in (b) and $\updownarrow$ in (c), respectively. Hence we have:

$$\Gamma \cdot y\!:\!\langle\tau_1,\ \tau_2\rangle \Vdash_{s_1' \sqcap s_2'} P_1 \mid P_2 \triangleright A_1 \odot A_2 \otimes B$$

where $B = \mathtt{p}y \otimes C$ with $\mathtt{p} \in \{\updownarrow, !, \Updownarrow\}$ or $B = C$ with either $C = \emptyset$ or $C = \Updownarrow x$. Therefore by applying (Res) to the above, we obtain:

$$\Gamma \Vdash_{s_1' \sqcap s_2'} (\boldsymbol{\nu}\, y\!:\!\langle\tau_1,\ \tau_2\rangle)(P_1 \mid P_2) \triangleright A_1 \odot A_2 \otimes C$$

with $A_1 \odot A_2 \otimes C \leq_{\mathtt{G}} A_1 \odot A_2 \otimes \Updownarrow x$ and $s_1' \sqcap s_2' \leq s_1 \sqcap s_2$, as desired.

Non-linear Case: (2) (Bra)-(Sel): Assume that

$$x[(y_1\!:\!\tau_1).P_1 \,\&\, (y_2\!:\!\tau_2).P_2] \mid \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, y_1\!:\!\tau_3).P_3 \longrightarrow (\boldsymbol{\nu}\, y\!:\!\langle\tau_1,\ \tau_3\rangle)(P_1 \mid P_3)$$

under the typing $\Gamma \Vdash_{s_1 \sqcap s_2 \sqcap s_3} x[(y_1\!:\!\tau_1).P_1 \,\&\, (y_2\!:\!\tau_2).P_2] \mid \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, y_3\!:\!\tau_3).P_3 \triangleright A_1 \odot A_2 \odot A_3 \otimes \Updownarrow x$ with the following derivations:

$$\begin{array}{lll} \Gamma \cdot y\!:\!\tau_i \Vdash_{s_i'} P_i \triangleright ?A_i \otimes B_i & \text{with} & \vdash [\vec{\tau}_1 \& \vec{\tau}_2]^{\Downarrow}_{s_1 \sqcap s_2} \leq \Gamma(x) \text{ and } s_1 \sqcap s_2 \leq s_1' \sqcap s_2' \ (i = 1, 2) \\ \Gamma \cdot y\!:\!\tau_3 \Vdash_{s_3'} P_3 \triangleright ?A_3 \otimes B_3 & \text{with} & \vdash [\vec{\tau}_1 \oplus \vec{\tau}_2]^{\Uparrow}_{s_3} \leq \Gamma(x) \text{ and } s_3 \leq s_3' \end{array}$$

where $B_i \leq_{\mathtt{G}} \mathtt{p}_i y_i \otimes \Updownarrow x$ with $i = 1, 2$ and $B_3 \leq_{\mathtt{G}} \mathtt{p}_i y_1 \otimes \Updownarrow x$ and $(A_1 \odot A_2) \asymp A_3$. With the similar reasoning of the above, we can derive $s_1 \sqcap s_2 = s_3 = s_1 \sqcap s_2 \sqcap s_3$ and $\tau_1 \asymp \tau_3$. Hence by (subsumption) of Lemma 5.3 again, we have:

$$\Gamma \cdot y\!:\!\langle\tau_1,\ \tau_3\rangle \Vdash_{s_1'} P_1 \triangleright ?A_1 \otimes B_1 \quad \text{and} \quad \Gamma \cdot y\!:\!\langle\tau_1,\ \tau_3\rangle \Vdash_{s_3'} P_3 \triangleright ?A_3 \otimes B_3$$

Note that by Lemma 3.5, we have $A_1 \asymp A_3$. Then applying (Par), we have:

$$\Gamma \cdot y\!:\!\langle\tau_1,\ \tau_3\rangle \Vdash_{s_1' \sqcap s_3'} P_1 \mid P_3 \triangleright A_1 \odot A_3 \otimes B$$

where $B = \mathtt{p}y \otimes C$ with $\mathtt{p} \in \{\updownarrow, \mathbf{!}, \Updownarrow\}$ or $B = C$ with either $C = \emptyset$ or $C = \Updownarrow x$. Note that $\mathbf{?}A_1 \odot \mathbf{?}A_2 \leq_{\mathtt{G}} \mathbf{?}A_1 \odot \mathbf{?}A_2 \odot \mathbf{?}A_3$. Therefore by the similar reasoning as the above case, we have done.

Non-linear Case: (3) (Rec-In)-(Out): Assume that

$$(\mu X(\vec{x}).x_0(y : \tau_1).P_1)\langle\vec{x}\rangle \mid \overline{x_0}(\boldsymbol{\nu}\, y : \tau_2).P_2 \longrightarrow (\boldsymbol{\nu}\, y : \langle\tau_1,\ \tau_2\rangle)(P_1\{\mathcal{E}/X\} \mid P_2)$$

and $\mathcal{E} = \mu X(\vec{x}).x_0(y : \tau_1).P_1$ under the typing
$\Gamma \Vdash_{s_1 \sqcap s_2} (\mu X(\vec{x}).x_0(y : \tau_1).P_1)\langle\vec{x}\rangle \mid \overline{x}(\boldsymbol{\nu}\, y : \tau_2).P_2 \triangleright A_1 \odot A_2 \otimes \Updownarrow x_0$ with the following derivations:

$$\begin{array}{lll} \Gamma \cdot y : \tau_1 \cdot X : \vec{\alpha} \Vdash_{s'_1} P_1 \triangleright \mathbf{?}A_1 \otimes B_1 & \text{with} & \vdash (\tau_1)^{\Downarrow}_{s_1} \leq \Gamma(x_0) \\ \Gamma \cdot y : \tau_2 \Vdash_{s'_2} P_2 \triangleright \mathbf{?}A_2 \otimes B_2 & \text{with} & \vdash (\tau_2)^{\Uparrow}_{s_2} \leq \Gamma(x_0) \end{array}$$

Note that by (Var) rule, if X appears in P_1, we know $s'_1 \leq \mathsf{sec}(\Gamma(x_0))$. Hence we have $s'_1 = s_1$. Then by Lemma 5.4 (1), we have:

$$\Gamma \cdot y : \tau_1 \Vdash_{s'_1} P_1\{\mathcal{E}/X\} \triangleright \mathbf{?}A_1 \otimes B_1$$

Now by the same reasoning of (Non-linear (In)-(Out)), we can obtain the required result.

Non-linear Case: (4) (Rec-Bra)-(Sel): We also use Lemma 5.4 (1). The rest is similar with the above case.

Truly-linear Case: (1) (In)-(Out): Assume that

$$x(y : \tau_1).P_1 \mid \overline{x}(\boldsymbol{\nu}\, y : \tau_2).P_2 \longrightarrow (\boldsymbol{\nu}\, y : \langle\tau_1,\ \tau_2\rangle)(P_1 \mid P_2)$$

under the typing $\Gamma \Vdash_{s_1 \sqcap s_2} x(y : \tau_1).P_1 \mid \overline{x}(\boldsymbol{\nu}\, y : \tau_2).P_2 \triangleright \mathbf{?}A_1 \odot \mathbf{?}A_2 \otimes B_1 \odot B_2 \otimes \updownarrow x$ with the following derivations:

$$\begin{array}{lll} \Gamma \cdot y : \tau_1 \Vdash_{s_1} P_1 \triangleright \mathbf{?}A_1 \otimes C_1 & \text{with} & \vdash (\tau_1)^{\downarrow}_{s'_1} \leq \Gamma(x) \\ \Gamma \cdot y : \tau_2 \Vdash_{s_2} P_2 \triangleright \mathbf{?}A_2 \otimes C_2 & \text{with} & \vdash (\tau_2)^{\uparrow}_{s'_2} \leq \Gamma(x) \end{array}$$

where $C_i/y = \downarrow\!\uparrow B_i$ or $C_i = \downarrow\!\uparrow B_i^{-y}$, $A_1 \asymp A_2$ and $B_1 \asymp B_2$ (Note that we have no relationship between s'_i and s_i). By the subtyping rules and Lemma 3.1 (3) again, we can derive $\tau_1 \asymp \tau_2$. By applying (subsumption) of Lemma 5.3 to the above derivations, we have:

$$\Gamma \cdot y : \langle\tau_1,\ \tau_2\rangle \Vdash_{s'_1} P_1 \triangleright A_1 \otimes C_1 \quad \text{and} \quad \Gamma \cdot y : \langle\tau_1,\ \tau_2\rangle \Vdash_{s'_2} P_2 \triangleright A_2 \otimes C_2$$

By Proposition 4.1 (1), this time, we have the following cases for B_1 and B_2:

(a) τ_1 and τ_2 are nonlinear and we have: $C_i \leq_{\mathtt{G}} \Updownarrow y \otimes B_i$ $(i = 1, 2)$.
(b) $\mathsf{md}(\tau_i) = \mathbf{!}$, $\mathsf{md}(\tau_j) = \mathbf{?}$ with $C_i = \mathbf{!}y \otimes B_i$ and $C_j \leq_{\mathtt{G}} \mathbf{?}y \otimes B_j$ $(i \neq j,\ i, j \in \{1, 2\})$.
(c) $\mathsf{md}(\tau_i) = \uparrow$, $\mathsf{md}(\tau_j) \in \downarrow$ with $C_i = \uparrow y \to B_i$ and $C_j = \downarrow y \to B_j$ $(i \neq j,\ i, j \in \{1, 2\})$.

In each case, we can check $C_1 \asymp C_2$, and $C_1 \odot C_2 = B_1 \odot B_2 \otimes \Updownarrow x$ or $C_1 \odot C_2 = B_1^{-x} \odot B_2^{-x}$ in (a), $C_1 \odot C_2 = B_1 \odot B_2 \otimes \mathbf{!}x$ in (b), and $C_1 \odot C_2 = B_1 \odot B_2 \otimes \updownarrow x$ in (c). Hence applying (Res) again, we have the required result (note that the security level does not change by this reduction).

Truly-linear Case: (2) (Bra)-(Sel): The only difference from the previous case is that we need to take care the security level. Assume $x[(y_1 : \tau_1).P_1 \,\&\, (y_2 : \tau_2).P_2] \mid \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, y_1 : \tau_3).P_3 \longrightarrow (\boldsymbol{\nu}\, y : \langle\tau_1,\ \tau_3\rangle)(P_1 \mid P_3)$ under the typing $\Gamma \Vdash_{s_1 \sqcap s_2 \sqcap s_3} x[(y_1 : \tau_1).P_1 \,\&\, (y_2 : \tau_2).P_2] \mid \overline{x}\mathtt{inl}(\boldsymbol{\nu}\, y_3 : \tau_3).P_3 \triangleright A_1 \odot A_2 \odot A_3 \otimes \Updownarrow x$ with the following derivations:

$$\begin{array}{lll} \Gamma \cdot y : \tau_i \Vdash_{s_i} P_i \triangleright \mathbf{?}A_i \otimes C_i & \text{with} & \vdash [\tau_1 \& \tau_2]^{\downarrow}_{s} \leq \Gamma(x) \text{ and } \mathsf{sec}(\Gamma(x)) \leq s_1 \sqcap s_2 \ (i = 1, 2) \\ \Gamma \cdot y : \tau_3 \Vdash_{s'_3} P_3 \triangleright \mathbf{?}A_3 \otimes C_3 & \text{with} & \vdash [\tau_3 \& \tau_4]^{\uparrow}_{s_3} \leq \Gamma(x) \text{ and } s_3 = \mathsf{sec}(\Gamma(x)) \sqcap s'_3 \ (i = 1, 2) \end{array}$$

where C_i is in the same condition of the above by replacing y to y_i with $i = 1, 2$ and y to y_1 if $i = 3$. Then by $\mathsf{sec}(\Gamma(x)) \leq s_1 \sqcap s_2$ and $s_3 = \mathsf{sec}(\Gamma(x)) \sqcap s'_3$, we can calculate that

$$s_1 \sqcap s_2 \sqcap s_3 \leq s_1 \sqcap s_3 \leq s_3 = \mathsf{sec}(\Gamma(x)) \sqcap s'_3 \leq s_1 \sqcap s_2 \sqcap s'_3 \leq s'_3 \sqcap s_1 .$$

By applying (Par) and (Res) as the previous cases, we have the desirable result.

Rec-Non-innocuous Case: (1) (In)-(Out): Assume that

$$(\mu X(\vec{x}).x_0(y\!:\!\tau_1).P_1)\langle\vec{x}\rangle \mid \overline{x_0}(\boldsymbol{\nu}\, y\!:\!\tau_2).P_2 \longrightarrow (\boldsymbol{\nu}\, y\!:\!\langle\tau_1,\ \tau_2\rangle)(P_1\{\mathcal{E}/X\} \mid P_2)$$

with $\mathcal{E} = \mu X(\vec{x}).x_0(y\!:\!\tau_1).P_1$ under the typing
$\Gamma \Vdash_{s_1 \sqcap s_2} (\mu X(\vec{x}).x_0(y\!:\!\tau_1).P_1)\langle\vec{x}\rangle \,|\, \overline{x}(\boldsymbol{\nu}\, y\!:\!\tau_2).P_2 \triangleright A_1 \odot A_2 \otimes \mathbf{!}x_0$ with the following derivations:

$$\begin{array}{ll} \Gamma \cdot y\!:\!\tau_1 \cdot X\!:\!\vec{\alpha} \Vdash_{s_1} P_1 \triangleright \mathbf{?}A_1 \otimes B_1 \otimes X\langle x_0\vec{w}\rangle & \text{with} \quad \vdash (\tau_1)^{\mathbf{!}}_{s_1,\boldsymbol{\mu}} \leq \Gamma(x_0) \\ \Gamma \cdot y\!:\!\tau_2 \Vdash_{s'_2} P_2 \triangleright \mathbf{?}A_2 \otimes B_2 & \text{with} \quad \vdash (\tau_2)^{\mathbf{?}}_{s_2,\boldsymbol{\mu}} \leq \Gamma(x_0) \end{array}$$

where $\vdash \alpha_i \leq \Gamma(x_i)$ and $B_1 \leq_{\mathtt{G}} \mathtt{p}_1 y$ for the input case, and $s_2 = s'_2 \sqcap \mathsf{sec}(\Gamma(x_0))$ and $B_2 \leq C \otimes \mathbf{?}x_0$ with $C/y = \downarrow\uparrow B$ or $C = \downarrow\uparrow B$ for the output case. Note that C takes the same form as C_i in (Truly-Linear (In)-(Out)). Hence, by Lemma 5.4 (2), we have:

$$\Gamma \cdot y\!:\!\tau_1 \Vdash_{s_1} P_1\{\mathcal{E}/X\} \triangleright \mathbf{?}A_1 \otimes B'_1 \otimes \mathbf{!}x_0$$

where either $B'_1 = B_1$ or $B'_1 = \mathbf{?}y$ with $B_1 = \mathbf{?}^{\iota}y$. Then by the similar reasoning of the above cases, we can easily check if $\mathtt{p}y \in B'_1 \odot B_2$, then with $\mathtt{p} \in \{\mathbf{!}, \Updownarrow, \downarrow\}$. Hence as similar to the previous case, applying (Res) and calculating security index as the above case, we can obtain the desired result.

Rec-Non-innocuous Case: (2) (Bra)-(Sel): We use Lemma 5.4 (3) by applying the similar reasoning of the above case.

Rec-Innocuous Case: (1) (In)-(Out): For action types, this is a special case $B'_1 = B_1$ of the proof of (Rec-Innocuous Case, (In)-(Out)) (here we use (4) of Lemma 5.4 instead of (2) in the previous case). The calculation of the security index is just similar to (Truly Linear: (In)-(Out)).

Rec-Innocuous Case: (2) (Bra)-(Sel): Similar to the above case and (Truly Linear: (Bra)-(Sel)).

E.4. Proofs for Section 6

E.4.1 Proof of Proposition 6.1

For both the typing and the subtyping relation, we check the "if" and the " only if" part, thus yielding four cases.

1. Typing ⇒. A straightforward induction on the structure of the derivation tree. Two rules deserve notice.

Case the last rule is (ASSIGN). Use rule (`assign`) followed by (`subs`); for the subtyping relation ion rule (`subs`), use our second rule.

Case the last rule is (SUBTYPE). For command, use our (`subs`) rule. For expressions, the subtyping $\rho_1 \subseteq \rho_2$ must be propagated through `and`-nodes towards the leaves. At the leafs, our (`var`) and (`bool`) rules take care of the matter.

For the system of threads with object map O, we must interpret the original rule [50] as saying that, for a system of threads O, all processes in dom(O) are typed with the same type. Then we type a system of n-processes with $n - 1$ applications of rule (PARALLEL).

2. Typing ⇐. A straightforward induction on the structure of the derivation tree.

3. Subtyping ⇒. Rule (BASE) follows directly from the encoding. Rule (REFLEX) branches into three cases: non-command phrase types follow from $\bot \leq \top$; command phrase H `cmd` follows from our second rule; and command phrase L `cmd` corresponds to our third rule. Finally Rule (CMD^-) corresponds to our second rule.

4. Subtyping ⇐. Our first and third rule corresponds to rule (REFLEX); our second rule corresponds to rule (CMD^-).

E.4.2 Proof of Proposition 6.2

For each of the directions of the proposition we analyze the three possible cases; each case corresponds to a subtyping rule in Figure 3, so there is not much to say. Nevertheless here is a detailed account.

$\Rightarrow$. (1) Pick $\rho = s\ \mathtt{cmd}_{\Downarrow}$, and $\rho = s'\ \mathtt{cmd}_{\Downarrow}$. To conclude that $s \geq s'$ use the first subtyping rule in Figure 3. (2) Pick $\rho = s\ \mathtt{cmd}_{\Uparrow}$, and $\rho = s'\ \mathtt{cmd}_{\Downarrow}$. To conclude that $s \geq s'$ use the second subtyping rule in Figure 3. (3) Pick $\rho = s\ \mathtt{cmd}_{\Uparrow}$, and $\rho = s'\ \mathtt{cmd}_{\Uparrow}$. To conclude that $s = s'$ use the third subtyping rule in Figure 3.

$\Leftarrow$. (1) Consider $\rho = s\ \mathtt{cmd}_{\Downarrow}$, and $\rho = s'\ \mathtt{cmd}_{\Downarrow}$, with $s \geq s'$. To show that $\rho \leq \rho'$ use the first subtyping rule in Figure 3. (2) Consider $\rho = s\ \mathtt{cmd}_{\Uparrow}$, and $\rho = s'\ \mathtt{cmd}_{\Downarrow}$, with $s \geq s'$. To show that $\rho \leq \rho'$ use the second subtyping rule in Figure 3. (3) Consider $\rho = s\ \mathtt{cmd}_{\Uparrow}$, and $\rho = s'\ \mathtt{cmd}_{\Uparrow}$, with $s = s'$. To show that $\rho \leq \rho'$ use the third subtyping rule in Figure 3.

E.4.3 The Link-lemma

This lemma is used three times in the proof of the Evaluation lemma 6.3.

Lemma E.7 (Link) *If* $\Gamma \cdot b : \mathtt{bool}^{!}_{s} \cdot b' : \mathtt{var}_{s'} \vdash_s P \triangleright ?A \otimes \langle\!\langle \tau \rangle\!\rangle_f \otimes !b \otimes ?^{\iota} b'$ *for* $s' \leq s$, *then* $\Gamma \cdot b' : \mathtt{var}_{s'} \vdash_s \mathbf{Link}\langle b^s, b'^{s'}, P\rangle \triangleright A \otimes \langle\!\langle \tau \rangle\!\rangle_f \otimes ?^{\iota} b'$.

PROOF: The derivation tree is of the form

$$\begin{array}{cc}
 & \text{Copy-cat} \\
\text{Hypothesis} & \text{5.3 (subsumption)} \\
\text{5.3 (weakening)} & \text{5.3 (weakening)} \\
\hline
\multicolumn{2}{c}{\text{Par}} \\
\multicolumn{2}{c}{\text{Res}}
\end{array}$$

Use the hypothesis with the security level of b equal to that of the sequent, namely s. Subsumption turns $b : \mathtt{bool}^{?}_{s}$ into $b : \mathtt{var}_s$. After the various applications of weakening, we have a typing of the form $\Gamma \cdot [\![E]\!] \cdot b : \mathtt{var}_s \cdot b' : \mathtt{bool}^{!}_{s}$. ■

E.4.4 The Evaluation lemma 6.3

By induction on the length of the derivation $E \vdash e : s$.

Case the last rule is (`var`) By hypothesis $E \vdash x : s$. Noticing that $x : \mathtt{var}_{\perp}$ is in $[\![E]\!]$, use the Read Proposition E.4.

Case the last rule is (`bool`) By hypothesis $E \vdash [\![\mathtt{b}]\!] : \perp$. Noticing that $[\![\mathtt{b}]\!] : \mathtt{var}_{\perp}$ is in $[\![E]\!]$, use the Link-Lemma E.7.

Case the last rule is (`and`) The derivation tree is of the form

$$\begin{array}{cc}
\text{Link-Lemma E.7} & \text{Link-Lemma E.7} \\
\text{Lemma 5.3 (weakening)} & \text{Lemma 5.3 (weakening)} \\
\text{Weak-}? & \text{Weak-}? \\
\hline
\multicolumn{2}{c}{\text{If-Lemma (Proposition E.1)}} \\
\multicolumn{2}{c}{\text{Induction hypothesis}} \\
\multicolumn{2}{c}{\text{Induction hypothesis}}
\end{array}$$

For each of the Link Lemmas, use the fact that $s_1 \leq s$ and $s_2 \leq s$. Weakening in each branch introduce the type and the action type of b_i: $\mathtt{bool}^{?}_{s_i}$ and $?^{\iota} b_i$, for $i = 1$ at the left, and $i = 2$ at the right.

E.4.5 Proof of Theorem 6.6

The proof for Theorem 6.4 is a special case of the present proof. The proof is by induction on the length of the derivation of $E \vdash c : \rho$.

Case the last rule is (`compose`). From the antecedent of the rule, we know that $E \vdash c_i : \rho$. Then we may use the induction hypothesis to get a derivation tree of the form below.

$$\dfrac{\begin{array}[b]{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (weakening)}\end{array}\quad\begin{array}[b]{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (weakening)}\\ \text{In}^{\downarrow}\text{ or In}\end{array}}{\begin{array}[t]{c}\text{Par}\\ \text{Res}\end{array}}$$

Weakening introduces in the basis the pair type $\langle\overline{[\![\rho]\!]},\ [\![\rho]\!]\rangle$ for g. Use rule In$^{\downarrow}$ when $\rho = s\,\mathtt{cmd}_{\Downarrow}$, and rule In when $\rho = s\ \mathtt{cmd}_{\Uparrow}$.

Case the last rule is (`parallel`).

$$\dfrac{\dfrac{\begin{array}[b]{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (weakening)}\end{array}\quad\begin{array}[b]{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (weakening)}\end{array}}{\text{Par}}\qquad\begin{array}{c}\text{Out}^{\uparrow}\\ \text{In}^{\downarrow}\\ \text{In}^{\downarrow}\end{array}\ \text{or}\ \begin{array}[b]{c}\text{Zero}\\ \text{Weak}^{\Updownarrow}\\ \text{Out}\\ \text{In}\\ \text{In}\end{array}}{\begin{array}[t]{c}\text{Par}\\ \text{Res}\\ \text{Res}\end{array}}$$

Weakening introduces in the basis the type $\langle\overline{[\![\rho]\!]},\ [\![\rho]\!]\rangle$ for both f_1 and f_2. Start at leaf Zero with a basis $[\![E]\!] \cdot f : [\![\rho]\!] \cdot f_1 : \langle\overline{[\![\rho]\!]},\ [\![\rho]\!]\rangle \cdot f_2 : \langle\overline{[\![\rho]\!]},\ [\![\rho]\!]\rangle$. Use rules Out$^{\downarrow}$, In$^{\downarrow}$, In$^{\downarrow}$ when $\rho = s\ \mathtt{cmd}_{\Downarrow}$, and rules Weak$^{\Updownarrow}$, Out, In, In when $\rho = s\ \mathtt{cmd}_{\Uparrow}$. Rule Weak$^{\Updownarrow}$ introduces $\Updownarrow f$ in the action type, as required by rule Out.

Case the last rule is (`assign`). Use the Write Proposition E.5, followed by the Eval Lemma 6.3.

Case the last rule is (`if`). From de antecendent of the rule, we know that $E \vdash c_i : \rho$. Applying the induction hypothesis, we get a derivation tree is of the form

$$\dfrac{\begin{array}[b]{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (weakening)}\\ \text{Weak-}\mathbf{?}\end{array}\quad\begin{array}[b]{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (weakening)}\\ \text{Weak-}\mathbf{?}\end{array}}{\begin{array}[t]{c}\text{Proposition E.1 (If)}\\ \text{Lemma 6.3 (Eval)}\end{array}}$$

Weakening in each branch introduce the type and the action type of y ($\mathtt{bool}^{\mathbf{?}}_{s}$ and $\mathbf{?}^{\iota}y$, respectively), thus placing the sequents in the form required by If-Proposition E.1.

Case the last rule is (`while`). From the antecedent of the rule we know that $E \vdash c : s\mathtt{cmd}_{\Uparrow}$. Use this fact for the induction hypothesis to get a derivation tree of the form

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\text{Induction hypothesis}\\ \text{Lemma 5.3 (subsumption)}\\ \text{Lemma 5.3 (weakening)}\end{array}\quad\begin{array}{c}\text{Var}\\ \text{Lemma 5.3 (subsumption)}\end{array}}{\begin{array}{c}\text{Par}\\ \text{Lemma 5.3 (weakening)}\\ \text{Weak-?}\end{array}}\quad\begin{array}{c}\text{Zero}\\ \text{Weak-}\Updownarrow\\ \text{Out}\\ \text{Weak-?}^{n}\end{array}}{\begin{array}{c}\text{Proposition E.1 (If)}\\ \text{Lemma 6.3 (Eval)}\\ \text{In}\\ \text{Rec}\\ \text{Par}\end{array}}\quad\begin{array}{c}\text{Zero}\\ \text{Weak-}\Updownarrow\\ \text{Out}\end{array}}{\text{Res}}$$

Let β be the pair type $\langle \mathtt{sync}_s^{\Downarrow},\ \mathtt{sync}_s^{\Uparrow}\rangle$. The (subsumption) of Lemma 5.3 changes the $\mathtt{sync}_s^{\Uparrow}$ type of g into β. Then, weakening introduces in the basis the types for f and X, respectively: $\mathtt{sync}_s^{\Uparrow}$ and $\mathtt{sync}_s^{\Downarrow}\mathtt{sync}_s^{\Downarrow}\tilde{\alpha}$. Start at the left Var leaf with a basis Γ of the form $[\![E]\!] \cdot f : \mathtt{sync}_s^{\Downarrow} \cdot g : \beta \cdot X : \mathtt{sync}_s^{\Downarrow}\mathtt{sync}_s^{\Downarrow}\tilde{\alpha}$. Below PAR, weakening introduces the type and the action type of y: $\mathtt{bool}_s^{?}$ and $?^{\iota}y$, respectively. It should then be clear that we must start at the left Zero leaf with a basis $\Gamma \cdot y : \mathtt{bool}_s^{?}$. Subsumption then changes the type of f into β. The $n+2$ Weak rules below (n is the number of variables in E) yield a sequent with an action type of the form $\langle\!\langle E\rangle\!\rangle \otimes \Updownarrow f \otimes \Updownarrow g$, thus preparing things to apply the If-Proposition E.1. Start at the right Zero leaf with a basis $[\![E]\!] \cdot f : \mathtt{sync}_s^{\Downarrow} \cdot g : \beta$. Rule Weak-$\Updownarrow$ below adds $\Updownarrow g$ to the empty action type, thus placing the sequent in the right form for rule Out. Notice that all the secrecy levels involved are s, the secrecy level of the while command.

Case the last rule is (`subs`). First notice that the original (`subs`) rule can be decomposed into two rules:

$$(\mathtt{subs\text{-}1})\,\frac{E \vdash c : s\ \mathtt{cmd}_{\Downarrow} \quad s' \leq s}{\vdash c : s'\ \mathtt{cmd}_{\Downarrow}} \qquad\qquad (\mathtt{subs\text{-}2})\,\frac{E \vdash c : s\ \mathtt{cmd}_{\Downarrow} \quad s' \leq s}{\vdash c : s\ \mathtt{cmd}_{\Uparrow}}$$

Then we have two subcases.

Subcase the last rule is (`subs-1`). Use the induction hypothesis followed by degradation (rule Deg_s).

Subcase the last rule is (`subs-2`). Noting that security levels are arbitrary for linear types, and using degradation, we know from the induction hypothesis that

$$[\![E]\!] \cdot f : \mathtt{sync}_{s'}^{\Uparrow} \vdash_{s'} [\![E \vdash c : \rho]\!]_f \triangleright \overline{\langle\!\langle E\rangle\!\rangle} \otimes \uparrow f.$$

We may assume without loss of generality that the rule before is not subsumption. Then we analyse all cases above to conclude that they work when f is linear as well as non-linear. Analysing the typing rules in figure 3, we see that potential complication may arise from (`assign`). For this case, notice that the Write Proposition E.5 handles both the linear and the non-linear case for f.

References

1. Abadi, M., Secrecy by typing in security protocols. *TACS'97*, LNCS, 611-637, Springer, 1997.

2. Abadi, M., Banerjee, A., Heintze, N. and Riecke, J., A core calculus of dependency, *POPL'99*, ACM, 1999.

3. Abadi, M., Fournet, C. and Gonthier, G., Secure Communications Processing for Distributed Languages. *LICS'98*, 868–883, IEEE, 1998.

4. Abadi, M. and Gordon, A.D., A calculus for cryptographic protocols: The spi calculus. Proc. Computer and Communication Security, 36–47, ACM, 1997.

5. Abramsky, S., Computational Interpretation of Linear Logic. *TCS*, vol 111, 1993.

6. Abramsky, S., Honda, K. and McCusker, G., Fully Abstract Game Semantics for General References, *LICS'98*, IEEE, 1998.

7. Abramsky, S., Jagadeesan, R. and Malacaria, P., Full Abstraction for PCF, 1994. To appear in *Info. & Comp.*

8. Berger, M, Honda, K., Vasconcelos, V. and Yoshida, N. *A Security-Sensitive Bisimilarity for a Typed π-Calculus.* To appear in QMW technical report, 2000.

9. Bodei, C, Degano, P., Nielson, F., and Nielson, H. Control Flow Analysis of π-Calculus. *CONCUR'98*, LNCS 1466, 84–98, Springer, 1998.

10. Boreale, M., De Nicola, R. and Pugliese, R., Proof Techniques for Cryptographic Processes, *LICS'99*, IEEE, 1999.

11. Boudol, G., The pi-calculus in direct style, *POPL'97*, 228–241, ACM, 1997.

12. Denning, D. and Denning, P., Certification of programs for secure information flow. *Communication of ACM*, ACM, 20:504–513, 1997.

13. Fiore, M. and Honda, K. Recursive Types in Games: Axiomatics and Process Representation. *LICS'98*, 1998.

14. Focardi, R. and Gorrieri, R., The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering,* 23(9), 1997.

15. Gay, S. and Hole, M., Types and Subtypes for Client-Server Interactions, *ESOP'99*, LNCS 1576, 74–90, Springer, 1999.

16. Girard, J.-Y., Linear Logic, *TCS*, Vol. 50, 1–102, 1987.

17. Goguen, J. and Meseguer, J., Security policies and security models. *IEEE Symposium on Security and Privacy*, 11–20, IEEE, 1982.

18. Gong, L., Prafullchandra, H. and Shcemers, R., Going beyond sandbox: an overview of the new security architecture in the Java development kit 1.2. USENIX Symposium on Internet Technologies and Systems, 1997.

19. Gray, J., Probabilistic interference. *Symposium on Security and Privacy*, 170–179, IEEE, 1990.

20. Heintze, N. and Riecke, J., The SLam calculus: programming with secrecy and integrity, *POPL'98*, 365–377, ACM, 1998.

21. Hennessy, M., Algebraic Theory of Processes. MIT Press, 1990.

22. Hennessy, M. and Riely, J., The security π-calculus, a talk in FCT99, 1999.

23. Hoare, C.A.R. Communicating Sequential Processes. Prentice Hall, 1985.

24. Honda, K., Types for Dyadic Interaction. *CONCUR'93*, LNCS 715, 509-523, 1993.

25. Honda, K., Composing Processes, *POPL'96*, 344-357, ACM, 1996.

26. Honda, K., A Theory of Types for the π-calculus. Typescript, 113 pp, October, 1999. Available from: `http://www.dcs.qmw.ac.uk/~kohei/`.

27. Honda, K. Vasco, V, and Yoshida, N. *Secure Information Flow as Typed Process Behaviour* (extended abstract). 16pp, To appear in Proc. of European Symposium on Programming (ESOP) 2000, LNCS, Springer, 2000.

28. Honda, K. and Tokoro, M., An Object Calculus for Asynchronous Communication. *ECOOP'91*, LNCS 512, pp.133–147, Springer-Verlag 1991.

29. Honda, K. and Yoshida, N. On Reduction-Based Process Semantics. *TCS*. 151, 437-486, 1995.

30. Honda, K. and Yoshida, N. Game-theoretic analysis of call-by-value computation. *TCS*, 221 (1999), 393–456, 1999.

31. Hyland, M. and Ong, L., Pi-calculus, dialogue games and PCF, *FPCA'93*, ACM, 1995.

32. Kobayashi, N., Pierce, B., and Turner, D., Linear types and π-calculus, *POPL'96*, 358–371, 1996.

33. Lafont, Y., Interaction Nets, *POPL'90*, 95–108, ACM, 1990.

34. Larsen, K. and Skou, A. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

35. Lopes, L, Silva, F. and Vasconcelos, F. A Virtual Machine for the TyCO Process Calculus. *PPDP'99*, 244–260, LNCS 1702, Springer, 1999.

36. Lowe, G. Defining Information Flow. MCS technical report, University of Leicester, 1999/3, 1999.

37. McLean, J. Security models and information flow. *IEEE Symposium on Security and Privacy*, 1990.

38. Milner, R., *Communication and Concurrency*, Prentice-Hall, 1989.

39. Milner, R., Functions as Processes. *MSCS* 2(2), 119–146, 1992.

40. Milner, R., Parrow, J.G. and Walker, D.J., A Calculus of Mobile Processes, *Info. & Comp.* 100(1), pp.1–77, 1992.

41. Myers, A. and Liskov, B., A decentralized model for information flow control. *SOSP'97*, 129–142, ACM, 1997.

42. Palsberg, J. and ∅rbaek , J., Trust in the λ-Calculus. Journal of Functional Programming, 7(6):557–591, 1997.

43. Pierce, B and Sangiorgi.D, Typing and subtyping for mobile processes, *MSCS* 6(5):409–453, 1996.

44. Sangiorgi, D. π-calculus, internal mobility, and agent-passing calculi. *TCS*, 167(2):235–271, 1996.

45. Sabelfield, A. and Sand, D. A per model of secure information flow in sequential programs. *ESOP'99*, LNCS 1576, Springer, 1999.

46. Sabelfield, A. and Sand, D. Probabilistic Noninterference for Multi-threaded Programs. Typescript, 13pp. Presented at Dera/RHBNC Workshop, December 1999.

47. Schneider, S. Security properties and CSP. *Symposium on Security and Privacy*, 174–187, 1996.

48. Vasconcelos, V., Typed concurrent objects. *ECOOP'94*, LNCS 821, pp.100–117. Springer, 1994.

49. Vasconcelos, V. and Honda, K., Principal Typing Schemes in a Polyadic π-Calculus. *CONCUR'93*, LNCS 715, 1993.

50. Volpano, D. and Smith, G., *Secure information flow in a multi-threaded imperative language*, pp.355–364, *POPL'98*, ACM, 1998.

51. Volpano, D. and Smith, G., Probabilistic Noninterference in a concurrent language, *IEEE 11th Security Foundations Workshop*, 1998.

52. Volpano, D. and Smith, G., Language Issues in Mobile Program Security. to appear in LNCS, Springer, 1999.

53. Volpano, D., Smith, G. and Irvine, C., *A Sound type system for secure flow analysis.* J. Computer Security, 4(2,3):167–187, 1996.

54. Walker, D., Objects in the π-calculus. Information and Computation, 116(2), pp.253-271, 1995.

55. Yoshida, N. Graph Types for Mobile Processes. *FST/TCS'16*, LNCS 1180, pp.371–386, Springer, 1996. The full version as LFCS Technical Report, ECS-LFCS-96-350, 1996.