Information Flow Control in Cyclic Process Networks

Bas van den Heuvel
HKA Karlsruhe, Germany
University of Freiburg, Germany
University of Groningen, The Netherlands

Farzaneh Derakhshan
Illinois Institutie of Technology, Chicago, IL, USA

Stephanie Balzer
Carnegie Mellon University, Pittsburgh, PA, USA

Abstract

Protection of confidential data is an important security consideration of today's applications. Of particular concern is to guard against unintentional leakage to a (malicious) observer, who may interact with the program and draw inference from made observations. Information flow control (IFC) type systems address this concern by statically ruling out such leakage. This paper contributes an IFC type system for message-passing concurrent programs, the computational model of choice for many of today's applications such as cloud computing and IoT applications. Such applications typically either implicitly or explicitly codify protocols according to which message exchange must happen, and to statically ensure protocol safety, behavioral type systems such as session types can be used. This paper marries IFC with session typing and contributes over prior work in the following regards: (1) support of realistic cyclic process networks as opposed to the restriction to tree-shaped networks, (2) more permissive, yet entirely secure, IFC control, exploiting cyclic process networks, and (3) considering deadlocks as another form of side channel, and asserting deadlock-sensitive noninterference (DSNI) for well-typed programs. To prove DSNI, the paper develops a novel logical relation that accounts for cyclic process networks. The logical relation is rooted in linear logic, but drops the tree-topology restriction imposed by prior work.

2012 ACM Subject Classification Theory of computation → Linear logic; Security and privacy → Logic and verification; Theory of computation → Process calculi; Theory of computation → Type theory

Keywords and phrases Cyclic process networks, linear session types, logical relations, deadlock-sensitive noninterference

Digital Object Identifier 10.4230/LIPIcs.ECOOP.2024.40

Related Version *Extended Version*: `https://arxiv.org/abs/2407.02304` [27]

Funding *Bas van den Heuvel*: Supported in part by the Dutch Research Council (NWO) under project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software).
Stephanie Balzer: Supported in part by the Air Force Office of Scientific Research under award number FA9550-21-1-0385 (Tristan Nguyen, program manager). Any opinions, findings and conclusions or recommendations expressed here are those of the author(s) and do not necessarily reflect the views of the U.S. Department of Defense.

1 Introduction

Many of today's emerging applications and systems such as cloud computing and IoT applications are inherently *concurrent* and *message passing*. Message passing also enjoys popularity in mainstream languages such as Erlang, Go, and Rust. Similar to functional languages with the λ-calculus as their theoretical model, the model of message-passing

concurrent languages is the process calculus [31, 47, 48]. A program in this setting amounts to a number of *processes* connected by *channels*, which compute by exchanging messages along these channels, rather than by β-reductions or writing to and reading from shared memory. Messages may even include channels themselves, a feature supported in the π-calculus [49, 59] and referred to as *higher-order* message passing.

Originally untyped [49], the π-calculus has gradually been enriched with types to prescribe the kinds of messages that can be exchanged over a channel [59] and to assert correctness properties, such as deadlock freedom and data-race freedom [40, 38, 39, 42]. Following in these footsteps, *session types* [32, 33] were conceived to additionally express the *protocols* underlying the exchange. Session types rely on a *linear* treatment of channels to model the state transitions induced by a protocol, which was even substantiated by a Curry-Howard correspondence between the session-typed π-calculus and linear logic [9, 66, 67, 64, 45, 10, 46]. Session types based on linear logic enjoy strong properties, comprising not only race and deadlock freedom but also protocol fidelity.

Security is another correctness consideration arising from today's applications and systems. One security concern in particular is the protection of confidential information, by preventing unintentional leakage to a (malicious) observer, who may interact with the program and draw inference from made observations. Type systems for *information flow control (IFC)* rule out such leakage by type checking [65, 60, 57], given a lattice over security levels and the labeling of observables (e.g., output, locations, channels) with these levels. Well-typed programs then prevent "flows from high to low" and guarantee *noninterference*, i.e., that an observer cannot infer any secrets from made observations. To guarantee noninterference, advanced proof methods such as logical relations [54, 2, 22, 50, 37, 63] and bisimulations [44, 62, 61, 58] are used. If side channels [57], such as the termination channel, are present, then the literature distinguishes *progress-sensitive* noninterference (PSNI) from *progress-insensitive* noninterference (PINI), where the former only equates a divergent program run with another diverging one, whereas the latter equates a divergent program run with any other run [24].

Whereas the development of IFC type systems has been an active research field for decades for imperative and predominantly sequential languages, their exploration in a concurrent, message-passing setting has been more confined to typed process calculi [34, 35, 17, 25, 41, 68, 55] and multiparty session types [13, 11, 14, 12]. Only recently, IFC has been adopted for session types based on linear logic [20, 5]. The resulting type systems exploit the strong guarantees arising from linear logic, which in particular curtail the network of processes arising at runtime to a tree structure. However, many real-world application scenarios are precluded from an insistence on a tree structure, instead requiring support of *cyclic process networks*. Session type systems [6, 7, 19, 29, 30] that allow for cyclic process networks increase expressivity while remaining rooted in linear logic.

This paper scales IFC to cyclic process networks and contributes an IFC type system for an asynchronous π-calculus with linear session types. To prove that well-typed processes in the resulting language enjoy noninterference, we develop a novel logical relation. Our development was challenged by the possibility of *deadlocks* that can arise in cyclic process networks and that constitute another form of side channel. To rule out side-channel attacks due to deadlocks, we introduce the notion of *deadlock-sensitive noninterference (DSNI)*, which only equates a deadlocking program with another deadlocking one. Using our logical relation we prove that well-typed processes in our language enjoy DSNI (fundamental theorem).

Cyclic process networks also turn out to be beneficial for IFC, as they permit secure programs that are rejected by existing IFC type systems for linear session types [20, 5]. These are programs that exploit the possibility of setting up several channels – rather than just one channel – between two processes, to separate low-security from high-security communication.

Contributions. Our contributions are threefold:

1. An IFC session type system for an asynchronous π-calculus with support for cyclic process networks (Sec. 3), that satisfies protocol fidelity and communication safety (Thm. 3.12).
2. A logical relation that induces an equivalence between typed processes (Sec. 4), defining our notion of DSNI (Def. 4.9).
3. The main result that well typedness implies DSNI (Thm. 5.1 in Sec. 5), following from the fundamental theorem (Thm. 5.7).

Outline. In addition to the above contributions, Sec. 2 gently introduces the key ideas behind the developments in this paper, Sec. 6 discusses related work, and Sec. 7 concludes the paper. Important proofs are detailed in part; remaining details, proofs, and auxiliary definitions are given in the extended version of this paper [27].

2 Key Ideas

In this section, we discuss the key ideas behind our contributions.

2.1 Cyclic Process Networks Afford Flexible Information Flow Control

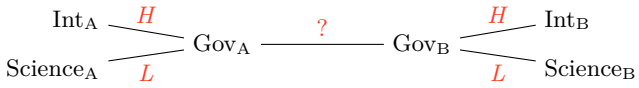
We motivate our developments through a high-level example, focusing on how cyclicity in process networks improves over prior works by increasing the flexibility of information flow control to support more realistic scenarios.

Collaborating governments. Consider two governments that want to collaborate on scientific and intelligence efforts. Clearly, the interactions between a government and an intelligence agency is confidential, whereas interactions between a government and the scientific community is not; intelligence may not leak to the scientific community, where there may be spies.

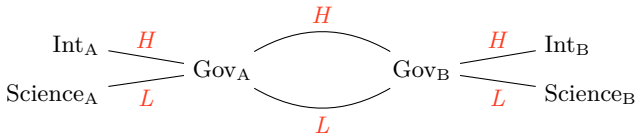
We make this more precise by establishing that information can be of *H*igh or *L*ow confidentiality, and that communication channels can be of *H*igh or *L*ow security. Clearly, information of *L*ow confidentiality can be transmitted over *H*igh-security channels, but not vice versa. We identify our two governments as $\mathrm{X} \in \{\mathrm{A}, \mathrm{B}\}$, each with departments (processes) $\mathrm{Gov}_\mathrm{X}, \mathrm{Int}_\mathrm{X}, \mathrm{Science}_\mathrm{X}$. We consider three scenarios, each of which connects these processes to form different process networks.

Scenario 1: No cyclicity. In Fig. 1a, the governments have only one channel to communicate on. Lines between departments denote communication channels, and their annotations indicate security levels. In this scenario, no processes are cyclically connected.

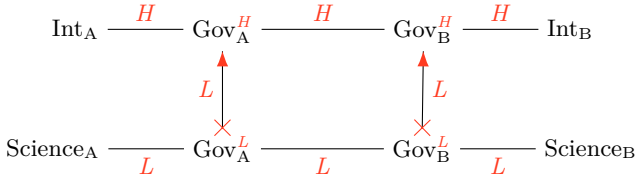
Notice that the channel connecting the two governments does not have a security level assigned. If we insist on intelligence exchange (i.e., on exchanging *H*igh-confidentiality information), this channel must be of *H*igh security. However, this inhibits scientific exchange: when a government receives information on a *H*igh-security channel, it cannot guarantee that the information is of *L*ow confidentiality, so it cannot share it with its scientific department over a *L*ow-security channel. Hence, the single channel of communication between the governments is unrealistic.



(a) Scenario 1: No cyclicity.



(b) Scenario 2: Doubly connected governments.



(c) Scenario 3: Extended cyclicity for more flexible IFC.

Figure 1 Collaborating governments: three scenarios.

Scenario 2: Doubly-connected governments. In Fig. 1b, we attempt to remedy this problem by adding a second channel of *L*ow security between the governments. Since the governments are connected on two separate channels, they are cyclically connected.

Now the governments can exchange scientific information and share this with their intelligence agencies. However, it is conceivable that a government makes decisions about which scientific information to share based on intelligence information. A clever spy may then be able to infer intelligence information from scientific information, *indirectly*. Hence, once a government receives intelligence information, it should refrain from sharing scientific information. Clearly, this scenario is still not realistic.

Scenario 3: Extended cyclicity for more flexible IFC. In Fig. 1c, we split our governments into *H*igh- and *L*ow-confidentiality departments. The *H*igh-confidentiality departments share intelligence information, and the *L*ow-confidentiality departments share scientific information. Crucially, the *L*ow-confidentiality department can share information with the *H*igh-confidentiality department, but not vice versa.

2.2 Threats to Noninterference due to Deadlocks

When process networks contain cyclic connections, there is a risk of *deadlocked* communication. For example, consider again the process network in Fig. 1b. Let us refer to the *H*igh-security channel between the governments as h, and the *L*ow-security channel between the governments as l. Suppose the two governments are implemented as follows (in pseudocode):

Gov_A := receive on h; send on l Gov_B := receive on l; send on h

The communication between the governments is deadlocked: each government is waiting to receive from the other, but the corresponding sends are blocked.

In process networks without cyclicity, this kind of deadlock does not occur. There are ways to prevent them through typing (cf., e.g., [19, 30, 7]), but the possible occurrence of deadlock introduced by cyclicity is realistic and a possible threat to noninterference. To see how, consider another pseudocode implementation, where s_A refers to the *L*ow-security channel between Gov_A and $Science_A$:

$$Gov_A := \text{receive x on h; if x == true then send on } s_A \text{ else deadlock} \quad (1)$$

In this scenario, a spy monitoring the information exchanged on s_A is indirectly able to infer *H*igh-confidentiality information: if information is sent on s_A, then the spy knows for sure that the value of x is true. This is why we are after IFC for noninterference that is *deadlock sensitive*.

2.3 IFC Type System in a Nutshell

In this paper, we implement IFC similarly as in previous works: by enriching a session type system with IFC annotations and requirements. We build on the session-typed asynchronous variant of the π-calculus of Van den Heuvel and Pérez [29, 30] (stripped from the "priority" mechanisms that rule out deadlock).

As anticipated in Sec. 2.1, channels are appointed *maximum-secrecy levels* (secrecy levels, for short), indicating the maximum secrecy of messages that can be sent on channels securely. For example, in Fig. 1b, the channel between Int_A and Gov_A has a secrecy level of *H*igh, while the channel between $Science_A$ and Gov_A has a secrecy level of *L*ow. This indicates that a spy with low-level security clearance can observe the messages sent on channels of secrecy level *L*ow but not *H*igh. A partial order on these secrecy levels forms a *secrecy lattice*; for example, $L \sqsubseteq H$ (*L*ow is lower than *H*igh).

As processes receive messages on channels, they learn "secrets", possibly influencing the information sent in future messages (referred to in the literature as *flow sensitivity* [57]). Key in our IFC is thus that it is forbidden to send messages on a channel if the level of secrecy learned so far exceeds the secrecy level of the channel. To ensure this, our type system assigns to each process a *running secrecy* that increases as the process receives higher-secrecy-level information. As processes evolve, the secrecy levels of their channels do not change, whereas their running secrecies do.

For example, in Fig. 1b, suppose Gov_A starts with a *L*ow running secrecy, thus being able to send messages to both Int_A and $Science_A$. After receiving a message from Int_A, the running secrecy of Gov_A becomes *H*igh: the secrecy level of the channel between them is *H*igh. Hence, after this message, Gov_A can no longer send messages to $Science_A$.

Finally, we need to address how our IFC handles deadlock sensitivity, as introduced in Sec. 2.2. It turns out that it is sufficient to rely on running secrecies and their dynamics as described above. To see how, consider again the implementation of Gov_A in (1). Assuming it starts with *L*ow running secrecy, the process receives on a *H*igh-security channel, so its running secrecy becomes *H*igh. Our IFC then disallows the process from sending on the *L*ow security channel. Hence, this example would be considered ill typed in our type system.

2.4 Logical Relation for DSNI in a Nutshell

Let us be more precise in what we mean by DSNI. A process may have a number of "unconnected" channels. By connecting these channels to other processes, we create a *context* in which to run the process. We refer to the channels connecting the process to its context as the *interface*. For example, in Fig. 1b, Gov_A can be considered as a standalone process, with the rest of the processes being a potential context for it. The interface is then the four channels connecting Gov_A to the other processes.

With DSNI, we assume the existence of an "attacker", a more precise definition of the "spy" mentioned in Sec. 2.1. This attacker knows the specification of our process, and has the ability to observe messages from and to the process over *observable channels*: channels in the interface that have secrecy levels up to a given secrecy level ξ. Moreover, the attacker cannot measure time but can observe the relative order in which messages are sent through different channels. By running our process in different contexts and observing the messages on observable channels, the attacker may be able to use its knowledge of the process' specification to infer information about messages on unobservable channels. As such, noninterference means that the attacker is not able to do so; in our case, we are after DSNI, because we do not want the attacker to infer information from deadlocks either.

In this paper, we define DSNI as an *equivalence* between the behavior on observable channels of the same process in different contexts. This equivalence is defined by means of a *logical relation*. The relation scrutinizes messages from and to the process on observable channels, and "ignores" messages on unobservable channels. Our main result is that well typedness implies DSNI (Thm. 5.1).

2.5 Technical Challenges

The subsequent sections first introduce our process language and then develop an IFC type system for that language and state and prove DSNI using a logical relation. These sections are naturally quite technical. To bridge the divide, we briefly survey here the main challenges our development had to overcome.

Asynchronous communication. Our process language is an *asynchronous* π-calculus with linear session types, based on Van den Heuvel and Pérez's Asynchronous Priority-based Classical Processes (APCP) [29, 30], but without recursion and "priority" mechanisms (which prevent deadlocks). As in the asynchronous π-calculus, outputs in our calculus do not have any continuations but are atomic processes composed in parallel with other processes. To model session sequencing, a process must adopt a *continuation-passing* style, in the sense that an output not only comprises a message but also a continuation channel.

When continuation channels are part of messages exchanged over an observable channel in the interface between a process and its context, the question comes up whether the continuation channel becomes observable as well. The natural impulse might be to consider them observable too. For sure this is the right choice in linear session-typed process calculi that confine process networks to trees [20, 5], guaranteeing that the continuation channel sent as part of the message resides within the sending process itself. However, due to the possibility of cycles in our setting, a continuation channel sent as part of a message may actually reside within the context outside the sending process. As a result, the logical relation has to consider the binding structure of the process and the context when determining observability of continuation channels. We detail this case analysis in Sec. 4 when we introduce the logical relation, with a pictorial illustration in Fig. 7.

Observable deadlocks. Deadlock-sensitive noninterference (DSNI) provides a very strong notion of noninterference in that it equates a deadlocking process only with another deadlocking one (as opposed to an arbitrary one). As a result, it prevents leakage through deadlocks, a side channel similar to the termination channel. DSNI is asserted by the definition of the logical relation and challenges the proof of Thm. 5.1, stating that well typedness implies DSNI. Thm. 5.1 is proved a generalized *fundamental theorem* (Thm. 5.7), which asserts that all executions of a process, if well typed, are related by the logical relation, up to the secrecy

level ξ of the observer. Because this theorem relates two different processes, but with the same observable behavior (where deadlocks are observable), the proof must maintain a tight correspondence between the two processes. This correspondence is achieved by employing the notion of *relevant nodes* (Def. 5.5), which are the parts of a process that can have observable outcomes either directly (by sending a message over the interface) or indirectly (by initiating a chain of messages ending with an observable one), and asserting that the relevant nodes of both processes are indistinguishable (up to structural congruence). Our notion of relevant nodes is inspired by Derakhshan et al. [20], but accounts for cyclic process networks.

Structural congruence and alpha equivalence. Our logical relation makes use of structural congruence to single out the action in a process producing an observable message. Because structural congruence permits alpha renaming, process relatedness must account for alpha-equivalence classes. As usual, proofs require a careful treatment of alpha renaming, which additionally becomes more nuanced by the existence of binders for observable names in contexts. This treatment becomes especially apparent in the so-called *catch-up* lemma (Lem. 5.10), a lemma used in the proof of the fundamental theorem to assert that two observably equivalent processes can "catch up" on each other's unobservable reductions.

3 Linear Session Types for Information Flow Control

In this section, we define our information flow control (IFC) type system. We first introduce our process language (an asynchronous π-calculus) along with a linear session-type system in Sec. 3.1. Then, we enrich the type system with IFC in Sec. 3.2. In Sec. 3.3, we prove that well-typed processes enjoy communication safety and protocol fidelity as corollaries of a type-preservation result. As we will see in Sec. 5, well typedness in the resulting IFC type system implies noninterference.

3.1 Process Language: Syntax, Semantics, and Types

Our process language is an asynchronous π-calculus, where parallel subprocesses communicate on connected channels. To be precise, we adapt the non-recursive fragment of Van den Heuvel and Pérez's Asynchronous Priority-based Classical Processes (APCP) [29, 30] by removing their "priority" mechanisms that prevent deadlock and adding our IFC.

Syntax. The syntactic elements of our language are typeset in a black and non-italic font. In our language, channels have two distinct endpoints, denoted $a, b, c, \ldots, x, y, z$ and further referred to as *names*. By design, all names are used linearly, meaning that they are used for a communication exactly once.

▶ **Definition 3.1** (Syntax). Processes $P, Q, R, \ldots$ *are defined by the following syntax:*

$$P, Q, R, \ldots ::= 0 \mid (P \mid Q) \mid (\nu xy)P \mid x[] \mid x(); P \mid x[b] \triangleleft j \mid x(z) \triangleright \{i : P_i\}_{i \in I} \mid x[a, b] \mid x(y, z); P$$

We write $P\{x/y\}$ to denote the capture-avoiding substitution of y for x in P. Process 0 denotes inaction. In $(P \mid Q)$, processes P and Q run in parallel; we often omit the parentheses. Restriction $(\nu xy)P$ binds x and y in P to form a channel, enabling communication.

Process $x[]$ closes the channel to which x belongs, and $x(); P$ waits for the channel to close before continuing as P. Selection $x[b] \triangleleft j$ sends the label j over x along with a name b; we refer to b as the selection's *continuation*, as it provides a means to continue communicating after the selection. Branch $x(z) \triangleright \{i : P_i\}_{i \in I}$ waits to receive on x a label $j \in I$ along with

a continuation b before continuing as $P_j\{b/z\}$; this binds z in each P_i. Send $x[a, b]$ sends names a and b over x; we typically refer to a and b as the send's payload and continuation, respectively, but there is no technical distinction between them. Receive $x(y, z); P$ waits to receive on x two names a and b before continuing as $P\{a/y, b/z\}$; this binds y and z in P. All names in a process are free unless bound as described above; we write $\mathrm{fn}(P)$ to denote the set of free names of P.

▶ **Example 3.2.** To illustrate process syntax, we further develop the example introduced in Sec. 2.1. We develop two simple accounts of $\mathrm{Gov_A}$: one where information flow is secure, and one where it is not.

In the first scenario, $\mathrm{Gov_A^L}$ passes a research outcome (oc) to $\mathrm{Gov_A^H}$, which determines a command for $\mathrm{Int_A}$:

$$\begin{aligned}
\mathrm{Gov_A^L} &:= (\nu \mathrm{a}_H^{1'} \mathrm{a}_H^1)(\mathrm{a}_H[\mathrm{a}_H^{1'}] \triangleleft \mathrm{oc}_2 \mid \mathrm{a}_H^1[]) \\
\mathrm{Gov_A^H} &:= \mathrm{a}_L(\mathrm{a}_L^1) \triangleright \left\{ \begin{array}{l} \mathrm{oc}_1 : (\nu \mathrm{a}_\mathrm{I}^{1'} \mathrm{a}_\mathrm{I}^1)(\mathrm{a}_\mathrm{I}[\mathrm{a}_\mathrm{I}^{1'}] \triangleleft \mathrm{act} \mid \mathrm{a}_L^1(); \mathrm{a}_\mathrm{I}^1[]), \\ \mathrm{oc}_2 : (\nu \mathrm{a}_\mathrm{I}^{1'} \mathrm{a}_\mathrm{I}^1)(\mathrm{a}_\mathrm{I}[\mathrm{a}_\mathrm{I}^{1'}] \triangleleft \mathrm{wait} \mid \mathrm{a}_L^1(); \mathrm{a}_\mathrm{I}^1[]) \end{array} \right\} \\
\mathrm{Int_A} &:= \mathrm{i_A}(\mathrm{i_A^1}) \triangleright \{\mathrm{act} : \mathrm{i_A^1}(); 0, \mathrm{wait} : \mathrm{i_A^1}(); 0\} \\
\mathrm{A_{secure}} &:= (\nu \mathrm{a}_H \mathrm{a}_L)(\nu \mathrm{a_I i_A})(\mathrm{Gov_A^L} \mid \mathrm{Gov_A^H} \mid \mathrm{Int_A})
\end{aligned}$$

In the second scenario, $\mathrm{Gov_A^H}$ receives intelligence (int) from $\mathrm{Int_A}$ and shares the information (inf) with $\mathrm{Gov_A^L}$:

$$\begin{aligned}
\mathrm{Int_A} &:= (\nu \mathrm{i_A^{1'} i_A^1})(\mathrm{i_A}[\mathrm{i_A^1}] \triangleleft \mathrm{int}_1 \mid \mathrm{i_A^1}[]) \\
\mathrm{Gov_A^H} &:= \mathrm{a_I}(\mathrm{a_I^1}) \triangleright \left\{ \begin{array}{l} \mathrm{int}_1 : (\nu \mathrm{a}_L^{1'} \mathrm{a}_L^1)(\mathrm{a}_L[\mathrm{a}_L^{1'}] \triangleleft \mathrm{inf}_1 \mid \mathrm{a}_\mathrm{I}^1(); \mathrm{a}_L^1[]), \\ \mathrm{int}_2 : (\nu \mathrm{a}_L^{1'} \mathrm{a}_L^1)(\mathrm{a}_L[\mathrm{a}_L^{1'}] \triangleleft \mathrm{inf}_2 \mid \mathrm{a}_\mathrm{I}^1(); \mathrm{a}_L^1[]) \end{array} \right\} \\
\mathrm{Gov_A^L} &:= \mathrm{a}_H(\mathrm{a}_H^1) \triangleright \{\mathrm{inf}_1 : \mathrm{a}_H^1(); 0, \mathrm{inf}_2 : \mathrm{a}_H^1(); 0\} \\
\mathrm{A_{insecure}} &:= (\nu \mathrm{a}_H \mathrm{a}_L)(\nu \mathrm{a_I i_A})(\mathrm{Gov_A^L} \mid \mathrm{Gov_A^H} \mid \mathrm{Int_A})
\end{aligned}$$

Variants of the π-calculus often include the forwarder process $[x \leftrightarrow y]$ which forwards any communications between x and y by fusing x and y. Here, we choose to omit forwarders for a smoother definition of our logical relation; they can be added as syntactic sugar using *identity expansion* (cf. the extended paper).

Semantics. The dynamics of our language is defined in terms of a reduction semantics, where each step represents the synchronization of complementary communications on the two endpoints of a channel. As usual, reduction relies on structural congruence, which restructures processes without affecting channel connections and the order of communications.

▶ **Definition 3.3** (Reduction Semantics). Structural congruence *is the least congruence on the syntax of processes (i.e., closed under arbitrary process contexts), denoted* $P \equiv Q$*, induced by the axioms in Fig. 2 (top).*

Reduction *is a binary relation on processes, denoted* $P \longrightarrow Q$*, defined by the rules in Fig. 2 (bottom). We write* $P \not\longrightarrow$ *to denote that there is no* Q *such that* $P \longrightarrow Q$*.*

Rule [SC-ALPHA] allows alpha conversion, i.e., renaming bound names. Rule [SC-PAR-NIL] defines 0 as the unit of parallel composition, and Rules [SC-PAR-SYMM] and [SC-PAR-ASSOC] define parallel composition as symmetric and associative, respectively. Rules [SC-RES-SYMM] and [SC-RES-ASSOC] define symmetry and associativity of restriction, respectively. Rule [SC-RES-COMM] defines commutativity of restriction, as long as this does not capture or free any names; this is often referred to as *scope extrusion*.

Structural congruence ($P \equiv Q$):

$$\frac{P \equiv_\alpha Q}{P \equiv Q}\ [\textsc{sc-alpha}] \qquad \frac{}{P \mid 0 \equiv P}\ [\textsc{sc-par-nil}] \qquad \frac{}{P \mid Q \equiv Q \mid P}\ [\textsc{sc-par-symm}]$$

$$\frac{}{(P \mid Q) \mid R \equiv P \mid (Q \mid R)}\ [\textsc{sc-par-assoc}] \qquad \frac{}{(\nu xy)P \equiv (\nu yx)P}\ [\textsc{sc-res-symm}]$$

$$\frac{}{(\nu xy)(\nu zw)P \equiv (\nu zw)(\nu xy)P}\ [\textsc{sc-res-assoc}] \qquad \frac{x, y \notin \mathrm{fn}(Q)}{(\nu xy)(P \mid Q) \equiv (\nu xy)P \mid Q}\ [\textsc{sc-res-comm}]$$

Reduction ($P \longrightarrow Q$):

$$\frac{}{(\nu xy)(x[] \mid y(); P) \longrightarrow P}\ [\textsc{red-close-wait}] \qquad \frac{j \in I}{(\nu xy)(x[b] \triangleleft j \mid y(w) \triangleright \{i : Q_i\}_{i \in I}) \longrightarrow Q_j\{b/w\}}\ [\textsc{red-sel-bra}]$$

$$\frac{}{(\nu xy)(x[a, b] \mid y(z, w); Q) \longrightarrow Q\{a/z, b/w\}}\ [\textsc{red-send-recv}] \qquad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}\ [\textsc{red-sc}] \qquad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}\ [\textsc{red-par}]$$

$$\frac{P \longrightarrow P'}{(\nu xy)P \longrightarrow (\nu xy)P'}\ [\textsc{red-res}]$$

Figure 2 Structural congruence (top) and reduction (bottom); cf. Def. 3.3.

Rules [RED-CLOSE-WAIT], [RED-SEL-BRA], and [RED-SEND-RECV] define synchronizations of complementary communications on names connected by restriction; these rules formalize the behavior described below Def. 3.1. Rules [RED-SC], [RED-PAR], and [RED-RES] close reduction under structural congruence, parallel composition, and restriction, respectively.

▶ **Example 3.4.** We illustrate process semantics on A_{secure} defined in Example 3.2. We have

$$\begin{aligned} A_{\text{secure}} &= (\nu a_H a_L)(\nu a_I i_A)(\mathrm{Gov}_A^L \mid \mathrm{Gov}_A^H \mid \mathrm{Int}_A) \\ &\equiv (\nu a_I i_A)((\nu a_H^{1'} a_H^1)((\nu a_H a_L)(a_H[a_H^{1'}] \triangleleft \mathrm{oc}_2 \mid a_L(a_L^1) \triangleright \{\ldots\}) \mid a_H^1[]) \mid \mathrm{Int}_A) \\ &\longrightarrow (\nu a_I i_A)((\nu a_H^{1'} a_H^1)((\nu a_I^{1'} a_I^1)(a_I[a_I^{1'}] \triangleleft \mathrm{wait} \mid a_H^{1'}(); a_I^1[]) \mid a_H^1[]) \mid \mathrm{Int}_A), \end{aligned}$$

from where asynchronous communication enables further communication between $a_H^{1'}$ and a_H^1 or between a_I and i_A; for example,

$$\longrightarrow (\nu a_I i_A)((\nu a_I^{1'} a_I^1)(a_I[a_I^{1'}] \triangleleft \mathrm{wait} \mid a_I^1[]) \mid \mathrm{Int}_A).$$

Types. We use linear session types to "tame" our processes. The system we use is derived from classical linear logic, so types are expressed as linear-logic propositions[1]; they are typeset in a blue and sans-serif font.

▶ **Definition 3.5** (Types). *Types* $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ *are defined by the following syntax:*

$$\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots ::= \mathsf{1} \mid \bot \mid \oplus\{i : \mathsf{A}\}_{i \in I} \mid \&\{i : \mathsf{A}\}_{i \in I} \mid \mathsf{A} \otimes \mathsf{B} \mid \mathsf{A} \parr \mathsf{B}$$

[1] This choice is usually motivated as it comes with deadlock freedom, but we have two different reasons: (1) it allows for direct compatibility with session-type systems for deadlock freedom, and (2) a logical basis gives us a very clean and well-behaved linear session type system, which we can carefully extend to serve our goals (here, guaranteeing noninterference by typing).

$$\text{[TYP-INACT]}\ \frac{}{\Omega \vdash 0 \mathbin{@} d :: \emptyset} \qquad \text{[TYP-PAR]}\ \frac{\Omega \Vdash d \sqsubseteq d_1' \sqcap d_2' \quad \Omega \vdash P \mathbin{@} d_1' :: \Gamma \quad \Omega \vdash Q \mathbin{@} d_2' :: \Delta}{\Omega \vdash P \mid Q \mathbin{@} d :: \Gamma, \Delta}$$

$$\text{[TYP-RES]}\ \frac{\Omega \vdash P \mathbin{@} d :: \Gamma, x : A[c], y : A^{\perp}[c]}{\Omega \vdash (\nu xy)P \mathbin{@} d :: \Gamma} \qquad \text{[TYP-CLOSE]}\ \frac{\Omega \Vdash d \sqsubseteq c}{\Omega \vdash x[] \mathbin{@} d :: x : 1[c]} \qquad \text{[TYP-WAIT]}\ \frac{\Omega \Vdash d' = d \sqcup c \quad \Omega \vdash P \mathbin{@} d' :: \Gamma}{\Omega \vdash x(); P \mathbin{@} d :: \Gamma, x : \bot[c]}$$

$$\text{[TYP-SEL]}\ \frac{\Omega \Vdash d \sqsubseteq c \quad j \in I}{\Omega \vdash x[b] \lhd j \mathbin{@} d :: x : \oplus\{i : A_i\}_{i\in I}[c], b : A_j^{\perp}[c]} \qquad \text{[TYP-BRA]}\ \frac{\Omega \Vdash d' = d \sqcup c \quad \forall i \in I.\ \Omega \vdash P_i \mathbin{@} d' :: \Gamma, z : A_i[c]}{\Omega \vdash x(z) \rhd \{i : P_i\}_{i\in I} \mathbin{@} d :: \Gamma, x : \&\{i : A_i\}_{i\in I}[c]}$$

$$\text{[TYP-SEND]}\ \frac{\Omega \Vdash d \sqsubseteq c}{\Omega \vdash x[a, b] \mathbin{@} d :: x : A \otimes B[c], a : A^{\perp}[c], b : B^{\perp}[c]}$$

$$\text{[TYP-RECV]}\ \frac{\Omega \Vdash d' = d \sqcup c \quad \Omega \vdash P \mathbin{@} d' :: \Gamma, y : A[c], z : B[c]}{\Omega \vdash x(y, z); P \mathbin{@} d :: \Gamma, x : A \parr B[c]}$$

Figure 3 Typing rules; cf. Def. 3.6.

Duality *is a unary operation on types, denoted* $A^{\perp}$*, defined as follows:*

$$1^{\perp} := \bot \qquad \oplus\{i : A_i\}_{i\in I}^{\perp} := \&\{i : A_i^{\perp}\}_{i\in I} \qquad A \otimes B^{\perp} := A^{\perp} \parr B^{\perp}$$
$$\bot^{\perp} := 1 \qquad \&\{i : A_i\}_{i\in I}^{\perp} := \oplus\{i : A_i^{\perp}\}_{i\in I} \qquad A \parr B^{\perp} := A^{\perp} \otimes B^{\perp}$$

Type 1 is associated with names that close channels, and $\bot$ with names that wait for channels to close. Types $\oplus\{i : A_i\}_{i\in I}$ and $\&\{i : A_i\}_{i\in I}$ are associated with names that make and expect labeled selections, respectively; given $j \in I$, A_j is the type of the continuation after j has been selected/received. Types $A \otimes B$ and $A \parr B$ are associated with names that send and receive, respectively; A and B are the types of the payload and continuation afterwards.

Duality is a key component of session types, as it defines precisely what is meant by complementary behavior; for example, $1^{\perp} = \bot$ is complementary to 1. Clearly, duality is an involution (i.e., $(A^{\perp})^{\perp} = A$).

Our type system is defined as a sequent calculus. In the following, ignore the annotations in *red and italic*; these annotations are for IFC, explained in Sec. 3.2.

▶ **Definition 3.6** (Type System)**.** Typing contexts $\Gamma, \Delta, \ldots$ *are defined by the following syntax:*

$$\Gamma, \Delta, \ldots ::= \emptyset \mid \Gamma, x : A[c]$$

Typing judgments *are denoted* $\Omega \vdash P \mathbin{@} d :: \Gamma$*. They are derived using the rules in Fig. 3.*

Typing contexts are thus sets of types assigned to names; the type system allows implicitly reordering these assignments in typing contexts. Whenever we write Γ, Δ, we assume that the sets of names appearing in Γ and Δ are disjoint.

Rule [TYP-INACT] types inaction under empty context. Rule [TYP-PAR] types parallel composition by splitting the typing context into disjoint parts, one for each parallel process. Rule [TYP-RES] types restriction by requiring the connected names to be dually typed. Rule [TYP-CLOSE] types a close with only its subject in the context. Dually, Rule [TYP-WAIT]

types a wait by removing its subject from the context of the continuation. Rule [TYP-SEL] types a selection; note that the continuation is typed dually to the continuation type of the selection itself, as this name will be received by a corresponding branch and used for further communications there. Dually, Rule [TYP-BRA] types a branch; it requires every continuation to be typed with the same context besides the type of the continuation name. Rule [TYP-SEND] types a send; the payload and continuation are typed dually, similar to the continuation in Rule [TYP-SEL]. Dually, Rule [TYP-RECV] types a receive.

▶ **Example 3.7.** To illustrate process typing, we type the secure variant of $\mathrm{Gov}_{\mathrm{A}}^{H}$ introduced in Example 3.2 as follows. We omit the *red and italic* IFC annotations entirely, as well as the typing of the oc_2 branch which is analogous to the oc_1 branch.

$$\dfrac{\dfrac{\dfrac{}{\vdash \mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{act} \;::\; \mathrm{a}_{\mathrm{I}} : \oplus\{act : 1, wait : 1\}, \mathrm{a}_{\mathrm{I}}^{1'} : \bot}\,[\text{TYP-SEL}] \quad \dfrac{\dfrac{}{\vdash \mathrm{a}_{\mathrm{I}}^{1}[] :: \mathrm{a}_{\mathrm{I}}^{1} : 1}\,[\text{TYP-CLOSE}]}{\vdash \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[] \;::\; \mathrm{a}_{L}^{1} : \bot, \mathrm{a}_{\mathrm{I}}^{1} : 1}\,[\text{TYP-WAIT}]}{\vdash \mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{act} \mid \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[] \;::\; \mathrm{a}_{L}^{1} : \bot, \mathrm{a}_{\mathrm{I}} : \oplus\{act : 1, wait : 1\}, \mathrm{a}_{\mathrm{I}}^{1'} : \bot, \mathrm{a}_{\mathrm{I}}^{1} : 1}\,[\text{TYP-PAR}]}{\vdash (\nu \mathrm{a}_{\mathrm{I}}^{1'} \mathrm{a}_{\mathrm{I}}^{1})(\mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{act} \mid \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[]) \;::\; \mathrm{a}_{L}^{1} : \bot, \mathrm{a}_{\mathrm{I}} : \oplus\{act : 1, wait : 1\}}\,[\text{TYP-RES}] \quad \vdots}{\vdash \mathrm{a}_{L}(\mathrm{a}_{L}^{1}) \triangleright \left\{ \begin{array}{l} \mathrm{oc}_1 : (\nu \mathrm{a}_{\mathrm{I}}^{1'} \mathrm{a}_{\mathrm{I}}^{1})(\mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{act} \mid \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[]), \\ \mathrm{oc}_2 : (\nu \mathrm{a}_{\mathrm{I}}^{1'} \mathrm{a}_{\mathrm{I}}^{1})(\mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{wait} \mid \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[]) \end{array} \right\} :: \begin{array}{l} \mathrm{a}_{L} : \&\{oc_1 : \bot, oc_2 : \bot\}, \\ \mathrm{a}_{\mathrm{I}} : \oplus\{act : 1, wait : 1\} \end{array}}\,[\text{TYP-BRA}]$$

3.2 Information Flow Control

We now enrich the type system presented thus far with IFC, such that well typedness guarantees noninterference (Sec. 5). That is, we introduce and explain the annotations in *red and italic* in Fig. 3, and formalize the intuititions given in Sec. 2.3.

We use $c, d, \ldots$ to denote secrecy levels. The relation between secrecy levels is denoted $c \sqsubseteq d$ (c is at most as secret as d), forming a lattice Ω. We write $\Omega \Vdash \phi$ to denote that the relation ϕ between secrecies holds within Ω. The least upper bound (join) and greatest lower bound (meet) are denoted $c \sqcup d$ and $c \sqcap d$, respectively.

Every name is assigned a secrecy level, denoted in typing contexts using square brackets after the name's type, as in $\mathrm{x} : \mathsf{A}[c]$. To remember the level of secrecy of the messages received by a process, we annotate the process in typing judgments with a *running secrecy*, denoted $\mathrm{P} \mathbin{@} d$. Given the running secrecy d of the process before an input and the secrecy level c of the input's subject, the input updates the running secrecy to the join $d \sqcup c$. When a process then performs an output, to make sure that the name is secured for handling the secrecy level of the outgoing message, our IFC requires that its running secrecy is not higher than that of the output's subject name.

We make these intuitions precise by discussing the IFC annotations on each typing rule in Fig. 3. Since Rule [TYP-INACT] does not involve communication, no secrecy checks are necessary. Rule [TYP-CLOSE] types an output, so it requires that the running secrecy d of the close is at most the secrecy level c of the closed name ($d \sqsubseteq c$): the information received so far (the running secrecy) is not more secret than the closed name. Rules [TYP-SEL] and [TYP-SEND] also type outputs, so their checks are similar. Rule [TYP-WAIT] types an input, so it sets the running secrecy d' of the continuation of the wait to the least upper bound of the running secrecy d before the wait and the secrecy level c of the name of the wait ($d' = d \sqcup c$; i.e., to the least secrecy level that is at least as high as both involved secrecies).

Rules [TYP-BRA] and [TYP-RECV] also type inputs, so they update running secrecies similarly. Rule [TYP-PAR] combines the running secrecies d'_1 and d'_2 of the parallel processes by taking a secrecy level d that is at most the greatest lower bound of d'_1 and d'_2 ($d \sqsubseteq d'_1 \sqcap d'_2$); this way, the parallel composition has a running secrecy of at most the least common secrecy level of its components, ensuring that each process in the composition has at least as much information as the parallel composition itself. Rule [TYP-RES] requires that the secrecies of the connected names coincide, ensuring that secrecy checks are consistent on both names of the created channel. Note that channels in typing contexts may have different secrecy levels. However, typing rules enforce that the channels in the same session (e.g., a send and its continuation) have the same secrecy levels. For example, Rule [TYP-SEL] ensures that channel x and its continuation b are both of the same secrecy level c.

▶ **Example 3.8.** To illustrate IFC in our type system, we consider again the typing of $\mathrm{Gov}_{\mathrm{A}}^{H}$ from Examples 3.2 and 3.7. We repeat the typing derivation and include IFC annotations, but omit processes and types to save space. Let Ω be the lattice with the only relation $L \sqsubset H$.

$$\dfrac{\dfrac{\dfrac{\dfrac{L \sqsubseteq H}{\Omega \vdash @\, L :: \mathrm{a_I} : [H], \mathrm{a_I^{1'}} : [H]}\,[\text{TYP-SEL}] \quad \dfrac{\dfrac{L \sqsubseteq H}{\Omega \vdash @\, L \sqcup L = L :: \mathrm{a_I^1} : [H]}\,[\text{TYP-CLOSE}]}{\Omega \vdash @\, L :: \mathrm{a}_L^1 : [L], \mathrm{a_I^1} : [H]}\,[\text{TYP-WAIT}]}{\Omega \vdash @\, L :: \mathrm{a}_L^1 : [L], \mathrm{a_I} : [H], \mathrm{a_I^{1'}} : [H], \mathrm{a_I^1} : [H]}\,[\text{TYP-PAR}]}{\Omega \vdash @\, L \sqcup L = L :: \mathrm{a}_L^1 : [L], \mathrm{a_I} : [H]}\,[\text{TYP-RES}] \quad \vdots}{\Omega \vdash @\, L :: \mathrm{a}_L : [L], \mathrm{a_I} : [H]}\,[\text{TYP-BRA}]$$

Hence, $\mathrm{Gov}_{\mathrm{A}}^{H}$ is considered secure in our type system, and as we will show in Sec. 5 this means that noninterference holds for this process.

However, the initial assignment of maximum secrecies to endpoints is chosen by the user. Well typedness and thus noninterference depends on this initial choice. To illustrate, reconsider the derivation above but now swapping the initial maximum secrecies:

$$\dfrac{\dfrac{\dfrac{\dfrac{H \not\sqsubseteq L}{\Omega \vdash @\, H :: \mathrm{a_I} : [L], \mathrm{a_I^{1'}} : [L]}\,[\text{TYP-SEL}] \quad \dfrac{\dfrac{H \not\sqsubseteq L}{\Omega \vdash @\, H \sqcup H = H :: \mathrm{a_I^1} : [L]}\,[\text{TYP-CLOSE}]}{\Omega \vdash @\, H :: \mathrm{a}_L^1 : [H], \mathrm{a_I^1} : [L]}\,[\text{TYP-WAIT}]}{\Omega \vdash @\, H :: \mathrm{a}_L^1 : [H], \mathrm{a_I} : [L], \mathrm{a_I^{1'}} : [L], \mathrm{a_I^1} : [L]}\,[\text{TYP-PAR}]}{\Omega \vdash @\, L \sqcup H = H :: \mathrm{a}_L^1 : [H], \mathrm{a_I} : [L]}\,[\text{TYP-RES}] \quad \vdots}{\Omega \vdash @\, L :: \mathrm{a}_L : [H], \mathrm{a_I} : [L]}\,[\text{TYP-BRA}]$$

This time, $\mathrm{Gov}_{\mathrm{A}}^{H}$ is not well typed: the IFC requirements of Rules [TYP-SEL] and [TYP-CLOSE] do not hold.

3.3 Type Preservation

Our type system guarantees by well typedness the usual correctness properties: *session fidelity* and *communication safety*. The former states that a process correctly implements the session types assigned to its names, and the latter that no communication mismatches take place (such as simultaneous outputs on both names of a channel).

Both these properties follow directly from *type preservation*: well typedness is preserved across structural congruences (subject congruence; Thm. 3.11) and reduction (subject reduction; Thm. 3.12). These results rely on two lemmas:

- Lem. 3.9 states that names that are not free in a process are not assigned in the typing of the process.
- Lem. 3.10 states that substitution in a process is reflected in its typing.

▶ **Lemma 3.9.** *Given* $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma$, *if* $\mathrm{x} \notin \mathrm{fn}(\mathrm{P})$, *then* $\mathrm{x} \notin \mathrm{dom}(\Gamma)$.

▶ **Lemma 3.10** (Substitution). *Given* $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma, \mathrm{x} : \mathsf{A}[c]$, *we have*

$$\Omega \vdash \mathrm{P}\{\mathrm{y}/\mathrm{x}\} \mathbin{@} d :: \Gamma, \mathrm{y} : \mathsf{A}[c].$$

▶ **Theorem 3.11** (Subject Congruence). *If* $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma$ *and* $\mathrm{P} \equiv \mathrm{Q}$ *for some* Q, *then* $\Omega \vdash \mathrm{Q} \mathbin{@} d :: \Gamma$.

Proof. By induction on the derivation of $\mathrm{P} \equiv \mathrm{Q}$. The inductive cases correspond to closure under arbitrary process contexts in Def. 3.1; these cases follow from the IH straightforwardly. The base cases correspond to the seven rules in Fig. 2 (top). In each case, we apply inversion on the typing of P to derive the typing of Q, and vice versa, with straightforward reasoning about running secrecies. The only interesting case is Rule [SC-PAR-ASSOC] $((\mathrm{P} \mid \mathrm{Q}) \mid \mathrm{R} \equiv \mathrm{P} \mid (\mathrm{Q} \mid \mathrm{R}))$, where we derive the running secrecy of $\mathrm{Q} \mid \mathrm{R}$ from that of $\mathrm{P} \mid \mathrm{Q}$, and vice versa. The full proof is in the extended paper. ◀

The following theorem states that (i) reduction preserves the well typedness of processes, and (ii) the running secrecy of processes may either stay the same or increase during reduction. This implies that a process never forgets the secrets it has learned, but it may learn more secrets as it reduces.

▶ **Theorem 3.12** (Subject Reduction). *If* $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma$ *and* $\mathrm{P} \longrightarrow \mathrm{Q}$ *for some* Q, *then* $\Omega \vdash \mathrm{Q} \mathbin{@} d' :: \Gamma$ *for some* d' *such that* $\Omega \Vdash d \sqsubseteq d'$.

Proof. By induction on the derivation of $\mathrm{P} \longrightarrow \mathrm{Q}$. The cases correspond to the reduction rules in Fig. 2 (bottom). In each case, we apply inversion on the typing of P to derive the typing of Q. The full proof is in the extended paper; here, we show the interesting case of Rule [RED-SEND-RECV]: $(\nu \mathrm{xy})(\mathrm{x}[\mathrm{a}, \mathrm{b}] \mid \mathrm{y}(\mathrm{z}, \mathrm{w}); \mathrm{P}) \longrightarrow \mathrm{P}\{\mathrm{a}/\mathrm{z}, \mathrm{b}/\mathrm{w}\}$. Given

$$\Omega \Vdash d \sqsubseteq d'_1 \sqcap d'_2, \tag{2}$$

$$\frac{\Omega \Vdash d'_1 \sqsubseteq c}{\Omega \vdash \mathrm{x}[\mathrm{a}, \mathrm{b}] \mathbin{@} d'_1 :: \mathrm{x} : \mathsf{A} \otimes \mathsf{B}[c], \mathrm{a} : \mathsf{A}^{\perp}[c], \mathrm{b} : \mathsf{B}^{\perp}[c]}\ [\textsc{typ-send}], \tag{3}$$

$$\Omega \Vdash d''_2 = d'_2 \sqcup c, \tag{4}$$

we have

$$\frac{\dfrac{(2) \quad (3) \quad \dfrac{(4) \quad \Omega \vdash \mathrm{P} \mathbin{@} d''_2 :: \Gamma, \mathrm{z} : \mathsf{A}^{\perp}[c], \mathrm{w} : \mathsf{B}^{\perp}[c]}{\Omega \vdash \mathrm{y}(\mathrm{z}, \mathrm{w}); \mathrm{P} \mathbin{@} d'_2 :: \Gamma, \mathrm{y} : \mathsf{A}^{\perp} \parr \mathsf{B}^{\perp}[c]}\ [\textsc{typ-recv}]}{\Omega \vdash \mathrm{x}[\mathrm{a}, \mathrm{b}] \mid \mathrm{y}(\mathrm{z}, \mathrm{w}); \mathrm{P} \mathbin{@} d :: \Gamma, \mathrm{x} : \mathsf{A} \otimes \mathsf{B}[c], \mathrm{y} : \mathsf{A}^{\perp} \parr \mathsf{B}^{\perp}[c], \mathrm{a} : \mathsf{A}^{\perp}[c], \mathrm{b} : \mathsf{B}^{\perp}[c]}\ [\textsc{typ-par}]}{\Omega \vdash (\nu \mathrm{xy})(\mathrm{x}[\mathrm{a}, \mathrm{b}] \mid \mathrm{y}(\mathrm{z}, \mathrm{w}); \mathrm{P}) \mathbin{@} d :: \Gamma, \mathrm{a} : \mathsf{A}^{\perp}[c], \mathrm{b} : \mathsf{B}^{\perp}[c]}\ [\textsc{typ-res}]$$

$$\Rightarrow \quad \frac{\text{Lem. 3.10 twice}}{\Omega \vdash \mathrm{P}\{\mathrm{a}/\mathrm{z}, \mathrm{b}/\mathrm{w}\} \mathbin{@} d''_2 :: \Gamma, \mathrm{a} : \mathsf{A}^{\perp}[c], \mathrm{b} : \mathsf{B}^{\perp}[c]}$$

By assumption and by definition, $\Omega \Vdash d''_2 \sqsupseteq d'_2$. Also, by definition, $\Omega \Vdash d'_2 \sqsupseteq d'_1 \sqcap d'_2$, so, by assumption, $\Omega \Vdash d'_2 \sqsupseteq d$. Hence, $\Omega \Vdash d''_2 \sqsupseteq d$. ◀

Liveness / Progress. Liveness / Progress properties specify the conditions under which processes can reduce. The progress property of APCP states that reduction takes place for a syntactic notion of "live" processes [30]. Since this result does not rely on APCP's priority mechanisms, it applies to our process language as well.

4 Logical Relation

This section defines an equivalence on typed processes up to "observable messages" (Def. 4.9) that we will use to state and prove DSNI in Sec. 5. We first give some preliminary definitions in Sec. 4.1, before defining the logical relation that induces this equivalence in Sec. 4.2.

4.1 Preliminary Definitions

As anticipated in Sec. 2.4, we are interested in the behavior of a process when it runs in different contexts. That is, we want to connect all the free names of the process in arbitrary ways. To this end, we define evaluation contexts: processes with a hole inside which a process may reduce (so under parallel composition and restriction). Evaluation contexts are typeset using an orange and monospaced font.

▶ **Definition 4.1** (Evaluation Context). Evaluation contexts *(*$\mathtt{E}$*) are defined as follows:*

$$\mathtt{E} ::= [\cdot] \mid \mathtt{E} \mid \mathtt{P} \mid (\nu \mathtt{xy})\mathtt{E}$$

We write $\mathtt{E}[\mathrm{P}]$ *to denote the process obtained by replacing the hole* $[\cdot]$ *in* $\mathtt{E}$ *by* P.

Any definitions on processes before and after this definition are lifted to evaluation contexts, without assigning any meaning to the hole. The exception is that alpha renaming does not apply to names that are bound by restriction but not free inside the scope of the restriction.

▶ **Example 4.2.** The following is an evaluation context:

$$\mathtt{E} := (\nu \mathtt{uw})(\nu \mathtt{xy})(\nu \mathtt{zv})(\mathtt{x}(); \mathtt{u}[] \mid \mathtt{z}[] \mid [\cdot])$$

Both u and w are bound in $\mathtt{E}$. Since u appears free within the scope of the restriction as the subject of a close, alpha renaming applies: $\mathtt{E} \equiv_\alpha (\nu \mathtt{aw})(\nu \mathtt{xy})(\nu \mathtt{zv})(\mathtt{x}(); \mathtt{a}[] \mid \mathtt{z}[] \mid [\cdot])$. However, the same does not hold for w: $\mathtt{E} \not\equiv_\alpha (\nu \mathtt{ua})(\nu \mathtt{xy})(\nu \mathtt{zv})(\mathtt{x}(); \mathtt{u}[] \mid \mathtt{z}[] \mid [\cdot])$.

We refer to the names that connect the process and its context as the *interface*. Our logical relation focuses on messages between context and process, i.e., messages that must pass through the interface. The following definition identifies outputs in process and context that are not blocked by prefixes. In particular, $\mathrm{aon}(\mathrm{P})$ is the set of names *along* which P is ready to output, and $\mathrm{acon}(\mathtt{E})$ is the set of names *to* which the context is ready to output.

▶ **Definition 4.3** (Active Interface Names). *We define the set of* active output names *of* P*, denoted* $\mathrm{aon}(\mathrm{P})$*, as the subjects of non-blocked outputs in* P*:*

$$\begin{array}{ccc} \mathrm{aon}(0) := \emptyset & \mathrm{aon}(\mathrm{P} \mid \mathrm{Q}) := \mathrm{aon}(\mathrm{P}) \cup \mathrm{aon}(\mathrm{Q}) & \mathrm{aon}((\nu \mathrm{xy})\mathrm{P}) := \mathrm{aon}(\mathrm{P}) \setminus \{\mathrm{x}, \mathrm{y}\} \\ \mathrm{aon}(\mathrm{x}[]) := \{\mathrm{x}\} & \mathrm{aon}(\mathrm{x}[\mathrm{a}, \mathrm{b}]) := \{\mathrm{x}\} & \mathrm{aon}(\mathrm{x}[\mathrm{b}] \lhd \mathrm{j}) := \{\mathrm{x}\} \\ \mathrm{aon}(\mathrm{x}(); \mathrm{P}) := \emptyset & \mathrm{aon}(\mathrm{x}(\mathrm{y}, \mathrm{z}); \mathrm{P}) := \emptyset & \mathrm{aon}(\mathrm{x}(\mathrm{z}) \rhd \{\mathrm{i} : \mathrm{P}_\mathrm{i}\}_{\mathrm{i} \in \mathrm{I}}) := \emptyset \end{array}$$

We define the set of active context output names *of* $\mathtt{E}$*, denoted* $\mathrm{acon}(\mathtt{E})$*, as the names in the interface of* $\mathtt{E}$ *that are connected to active output names of* $\mathtt{E}$ *through restriction:*

$$\mathrm{acon}(\mathtt{E}) := \{\mathrm{x} \mid \exists \mathrm{y}, \mathtt{E}'.\ \left(\mathtt{E} \equiv (\nu \mathrm{xy})\mathtt{E}' \wedge \mathrm{x} \notin \mathrm{fn}(\mathtt{E}') \wedge \mathrm{y} \in \mathrm{aon}(\mathtt{E}')\right)\}$$

We define the set of active interface names *of* $\mathtt{E}$ *and* P *as the union of the active context output names of* $\mathtt{E}$ *and the active output names of* P*:*

$$\mathrm{ain}(\mathtt{E},\mathrm{P}) := \mathrm{acon}(\mathtt{E}) \cup \mathrm{aon}(\mathrm{P})$$

▶ **Example 4.4.** We illustrate the active interface names between $\mathrm{P} := \mathrm{y}[] \mid \mathrm{w}(); \mathrm{v}(); 0$ and $\mathtt{E}$ from Example 4.2. It is easy to see that $\mathrm{aon}(\mathrm{P}) = \{\mathrm{y}\}$. To determine $\mathrm{acon}(\mathtt{E})$ we search for names in $\mathtt{E}$ that are bound by restriction to names used for output, but not used themselves. That is, we look for names in the interface between the context and the containing process, on which the containing process can expect to receive an output from the context. For example, in $\mathtt{E}$, name v is bound to z which is used for an output, while v itself is not used (it appears in the interface). Since there are no further such names, we have $\mathrm{acon}(\mathtt{E}) = \{\mathrm{v}\}$. As such, $\mathrm{ain}(\mathtt{E},\mathrm{P}) = \{\mathrm{y},\mathrm{v}\}$.

The interface is where an attacker (cf. Sec. 2.4) may observe the behavior of our process. In this, we assume that the attacker can only observe messages up to a certain secrecy level ξ. As such, our relation is only interested in the behavior of the process on *observable* channels. To this end, we define a projection on typing contexts to filter out unobservable channels. Also, we define when a process in context is well typed with respect to a given typing context of observable channels.

▶ **Definition 4.5** (Projection and Networks). *Given a secrecy lattice* Ω*, a secrecy level* $\xi \in \mathrm{dom}(\Omega)$*, and a typing context* Γ*, we define the* projection $\Gamma \Downarrow_\Omega \xi$ *as follows:*

$$(\Gamma, \mathrm{x} : \mathsf{A}[c]) \Downarrow_\Omega \xi := \begin{cases} (\Gamma \Downarrow_\Omega \xi), \mathrm{x} : \mathsf{A}[c] & \text{if } \Omega \Vdash c \sqsubseteq \xi \\ \Gamma \Downarrow_\Omega \xi & \text{if } \Omega \Vdash c \not\sqsubseteq \xi \end{cases} \qquad \emptyset \Downarrow_\Omega \xi := \emptyset$$

We often omit Ω *when it is clear from the context.*

We say $\mathtt{E}$ *and* P form a network with interface Γ observable up to ξ under Ω*, denoted* $(\mathtt{E},\mathrm{P}) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$ *if and only if there are* d, d', Γ' *such that* $\Gamma = \Gamma' \Downarrow_\Omega \xi$*,* $\Omega \vdash \mathrm{P} \mathbin{@} d' :: \Gamma'$*, and* $\Omega \vdash \mathtt{E}[\mathrm{P}] \mathbin{@} d :: \emptyset$*. By abuse of notation, we write* $(\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$ *to denote* $(\mathtt{E}_1, \mathrm{P}_1) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$ *and* $(\mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$*.*

▶ **Example 4.6.** We anticipate illustrating noninterference on the secure running example introduced in Example 3.2 on a *Low* secrecy channel, by considering the projection of its typing context and an evaluation context to form a network. Recall the typing and IFC annotations from Examples 3.7 and 3.8:

$$\vdash \mathrm{Gov}_{\mathrm{A}}^{H} \mathbin{@} L :: \Gamma' = \mathrm{a}_L : \&\{\mathsf{oc}_1 : \bot, \mathsf{oc}_2 : \bot\}[L], \mathrm{a}_{\mathrm{I}} : \oplus\{\mathsf{act} : 1, \mathsf{wait} : 1\}[H].$$

We have $\Gamma := \Gamma' \Downarrow L = \mathrm{a}_L : \&\{\mathsf{oc}_1 : \bot, \mathsf{oc}_2 : \bot\}[L]$. Let $\mathtt{E} := (\nu \mathrm{a}_H \mathrm{a}_L)(\nu \mathrm{a}_{\mathrm{I}} \mathrm{i}_{\mathrm{A}})(\mathrm{Gov}_{\mathrm{A}}^{L} \mid [\cdot] \mid \mathrm{Int}_{\mathrm{A}})$. It is straightforward to confirm that $\vdash \mathtt{E}[\mathrm{Gov}_{\mathrm{A}}^{H}] \mathbin{@} L :: \emptyset$. Hence, $(\mathtt{E}, \mathrm{Gov}_{\mathrm{A}}^{H}) \in \mathsf{Net}^{L}(\Gamma)$.

In our relation, we want to exhaust reductions on unobservable channels, after which we scrutinize behavior on observable names in the interface. To this end, we define *unobservable reductions*, which entail reductions internal to the process or the context, but also communications between process and context on unobservable interface channels.

▶ **Definition 4.7** (Unobservable Reduction). *We define* unobservable reduction *as*

$$\mathtt{E}, \mathrm{P} \longrightarrow_{\Omega;\xi;\Gamma} \mathtt{E}', \mathrm{P}'$$

if and only if $(\mathtt{E},\mathrm{P}) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$*,* $\mathtt{E}[\mathrm{P}] \longrightarrow \mathtt{E}'[\mathrm{P}']$ *and* $(\mathtt{E}',\mathrm{P}') \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$*. We write* $\longrightarrow^{?}_{\Omega;\xi;\Gamma}$ *(resp.* $\longrightarrow^{*}_{\Omega;\xi;\Gamma}$*) for the reflexive (resp. reflexive transitive) closure of* $\longrightarrow_{\Omega;\xi;\Gamma}$*, and* $\mathtt{E}, \mathrm{P} \not\longrightarrow_{\Omega;\xi;\Gamma}$ *to denote that there are no* $\mathtt{E}', \mathrm{P}'$ *such that* $\mathtt{E}, \mathrm{P} \longrightarrow_{\Omega;\xi;\Gamma} \mathtt{E}', \mathrm{P}'$*.*

$$(\mathtt{E}_1[\mathrm{P}_1];\mathtt{E}_2[\mathrm{P}_2]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma]\!] \iff \tag{5}$$

$$\wedge\, (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma) \tag{6}$$

$$\wedge\, \forall \mathtt{E}_1', \mathrm{P}_1'.\ \mathtt{E}_1, \mathrm{P}_1 \longrightarrow^*_{\Omega;\xi;\Gamma} \mathtt{E}_1', \mathrm{P}_1' \not\rightarrow_{\Omega;\xi;\Gamma} \tag{7}$$

$$\implies \exists \mathtt{E}_2', \mathrm{P}_2'.\ \mathtt{E}_2, \mathrm{P}_2 \longrightarrow^*_{\Omega;\xi;\Gamma} \mathtt{E}_2', \mathrm{P}_2' \not\rightarrow_{\Omega;\xi;\Gamma} \tag{8}$$
$$\wedge \forall \mathrm{x} \in \big(\mathrm{ain}(\mathtt{E}_1', \mathrm{P}_1') \cup \mathrm{ain}(\mathtt{E}_2', \mathrm{P}_2')\big) \cap \mathrm{dom}(\Gamma).\ (\mathtt{E}_1', \mathrm{P}_1'; \mathtt{E}_2', \mathrm{P}_2') \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma]\!]$$
$$\wedge\, \mathrm{aon}(\mathrm{P}_1') \cap \mathrm{dom}(\Gamma) = \mathrm{aon}(\mathrm{P}_2') \cap \mathrm{dom}(\Gamma)$$

Figure 4 Term interpretation.

Our relation often requires "zooming in" on specific parts of processes. To this end, we define notions to deal with atomic parts of processes.

▶ **Definition 4.8** (Nodes and Normal Forms). *Given a process* P*, we say* P *is a* node *if* $\mathrm{P} \not\equiv \mathrm{Q} \mid \mathrm{R}$*,* $\mathrm{P} \not\equiv (\nu \mathrm{xy})\mathrm{Q}$*, and* $\mathrm{P} \not\equiv 0$*.*

We say a process $\mathrm{Q} = (\nu \mathrm{x_i y_i})_{\mathrm{i}\in\mathrm{I}} \prod_{\mathrm{j}\in\mathrm{J}} \mathrm{P_j}$ *is in* normal form *if, for every* $\mathrm{j} \in \mathrm{J}$*,* $\mathrm{P_j}$ *is a node, and* Q *is a* normal form of P *if* $\mathrm{P} \equiv \mathrm{Q}$*. Normal forms are closed under structural congruence: every process induces an equivalence class of structurally congruent normal forms.*

Given a process in normal form $\mathrm{Q} = (\nu \mathrm{x_i y_i})_{\mathrm{i}\in\mathrm{I}} \prod_{\mathrm{j}\in\mathrm{J}} \mathrm{P_j}$*, we define* $\mathrm{nodes}(\mathrm{Q}) := \{\mathrm{P_j} \mid \mathrm{j} \in \mathrm{J}\}$ *and* $\mathrm{binders}(\mathrm{Q}) := \big\{\{\mathrm{x_i}, \mathrm{y_i}\} \mid \mathrm{i} \in \mathrm{I}\big\}$*. By abuse of notation, given a process* P *not necessarily in normal form, we write* $\mathrm{nodes}(\mathrm{P})$ *to denote* $\mathrm{nodes}(\mathrm{Q})$ *for an arbitrary normal form* Q *of* P*.*

For example, let

$$\mathrm{P} := (\nu \mathrm{uw})\big((\nu \mathrm{xy})(\mathrm{z}();\mathrm{x}[] \mid \mathrm{y}();\mathrm{u}[]) \mid \mathrm{w}();0\big) \mid 0 \quad \mathrm{Q} := (\nu \mathrm{uw})(\nu \mathrm{xy})(\mathrm{z}();\mathrm{x}[] \mid \mathrm{y}();\mathrm{u}[] \mid \mathrm{w}();0).$$

Then Q is a normal form of P, with $\mathrm{nodes}(\mathrm{Q}) = \{\mathrm{z}();\mathrm{x}[], \mathrm{y}();\mathrm{u}[], \mathrm{w}();0\}$ and $\mathrm{binders}(\mathrm{Q}) = \{\{\mathrm{u},\mathrm{w}\},\{\mathrm{x},\mathrm{y}\}\}$.

4.2 The Relation

Having presented all its ingredients, we now introduce our logical relation. As usual, the relation consists of two parts: a *term interpretation* and a *value interpretation*, defined by mutual multiset induction on the interfaces of processes. The term interpretation is the main part of the relation, and is responsible for calling on the value interpretation when a message is ready to be communicated across the observable interface, as well as ensuring deadlock sensitivity of our noninterference result. The value interpretation zooms in on the interface, and ensures that the two runs of the process behave identically on observable messages that are to be communicated across the interface.

Let us start by presenting our term interpretation, denoted $\mathsf{E}^{\Omega;\xi}[\![\Gamma]\!]$, in Fig. 4. It relates pairs of processes, given a secrecy lattice Ω, a secrecy level $\xi \in \mathrm{dom}(\Omega)$, and an interface Γ. We break down its definition part by part. Part (5) implicitly requires each process to be separable into a context $\mathtt{E}_\mathrm{i}$ and a process $\mathrm{P_i}$. Part (6) then requires the interface between $\mathtt{E}_\mathrm{i}$ and $\mathrm{P_i}$ to correspond to Γ up to observability ξ (cf. Def. 4.5). Part (7) exhausts unobservable reductions for $\mathtt{E}_1, \mathrm{P}_1$ (cf. Def. 4.7) in every way possible, resulting in $\mathtt{E}_1', \mathrm{P}_1'$. Part (8) first requires $\mathtt{E}_2, \mathrm{P}_2$ to "catch up" through exhaustive unobservable reductions, resulting in $\mathtt{E}_2', \mathrm{P}_2'$. Then, Part (8) invokes the value interpretation, presented next, to scrutinize any messages that are ready to be transferred across the observable part of the interface of either $\mathtt{E}_\mathrm{i}', \mathrm{P}_\mathrm{i}'$ (cf. Def. 4.3). Finally, Part (8) ensures deadlock sensitivity by requiring the observable messages

$$
\begin{array}{ll}
1 & (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma, \mathrm{x} : 1[c]]\!] \iff \big((\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma, \mathrm{x} : 1[c]) \\
& \quad \wedge \mathrm{P}_1 \equiv \mathrm{x}[] \mid \mathrm{P}_1' \wedge \mathrm{P}_2 \equiv \mathrm{x}[] \mid \mathrm{P}_2' \wedge (\mathtt{E}_1\left[\mathrm{x}[] \mid \mathrm{P}_1'\right]; \mathtt{E}_2\left[\mathrm{x}[] \mid \mathrm{P}_2'\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma]\!]\big) \\
\hline
\oplus & (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma, \mathrm{x} : \oplus\{\mathrm{i} : \mathsf{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}[c]]\!] \iff & (9) \\
& \quad (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma, \mathrm{x} : \oplus\{\mathrm{i} : \mathsf{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}[c]) & (10) \\
& \quad \wedge \exists \mathrm{j} \in \mathrm{I}.\ \mathrm{x}[\mathrm{b}_1] \triangleleft \mathrm{j} \in \mathrm{nodes}(\mathrm{P}_1) \wedge \mathrm{x}[\mathrm{b}_2] \triangleleft \mathrm{j} \in \mathrm{nodes}(\mathrm{P}_2) & (11) \\
& \quad \wedge \mathrm{b}_1 \in \mathrm{dom}(\Gamma) \implies \mathrm{b}_1 = \mathrm{b}_2 \wedge \mathrm{P}_1 \equiv \mathrm{x}[\mathrm{b}_1] \triangleleft \mathrm{j} \mid \mathrm{P}_1' \wedge \mathrm{P}_2 \equiv \mathrm{x}[\mathrm{b}_2] \triangleleft \mathrm{j} \mid \mathrm{P}_2' & (12) \\
& \qquad\qquad \wedge (\mathtt{E}_1\left[\mathrm{x}[\mathrm{b}_1] \triangleleft \mathrm{j} \mid \mathrm{P}_1'\right]; \mathtt{E}_2\left[\mathrm{x}[\mathrm{b}_2] \triangleleft \mathrm{j} \mid \mathrm{P}_2'\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma \setminus \mathrm{b}_1]\!] & \\
& \quad \wedge \mathrm{b}_1 \notin \mathrm{dom}(\Gamma) \implies \big(\mathrm{b}_2 \notin \mathrm{dom}(\Gamma) & (13) \\
& \qquad \wedge \mathrm{P}_1 \equiv (\nu \mathrm{b}_1\mathrm{b}')(\mathrm{x}[\mathrm{b}_1] \triangleleft \mathrm{j} \mid \mathrm{P}_1') \wedge \mathrm{P}_2 \equiv (\nu \mathrm{b}_2\mathrm{b}')(\mathrm{x}[\mathrm{b}_2] \triangleleft \mathrm{j} \mid \mathrm{P}_2') & \\
& \qquad \wedge (\mathtt{E}_1\left[(\nu \mathrm{b}_1\mathrm{b}')(\mathrm{x}[\mathrm{b}_1] \triangleleft \mathrm{j} \mid \mathrm{P}_1')\right]; \mathtt{E}_2\left[(\nu \mathrm{b}_2\mathrm{b}')(\mathrm{x}[\mathrm{b}_2] \triangleleft \mathrm{j} \mid \mathrm{P}_2')\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma, \mathrm{b}' : \mathsf{A}_\mathrm{j}[c]]\!]\big) & \\
\hline
\otimes & (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma, \mathrm{x} : \mathsf{A} \otimes \mathsf{B}[c]]\!] \iff & \\
& \quad (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma, \mathrm{x} : \mathsf{A} \otimes \mathsf{B}[c]) & \\
& \quad \wedge \mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \in \mathrm{nodes}(\mathrm{P}_1) \wedge \mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \in \mathrm{nodes}(\mathrm{P}_2) & \\
& \quad \wedge \mathrm{a}_1, \mathrm{b}_1 \in \mathrm{dom}(\Gamma) \implies \mathrm{a}_1 = \mathrm{b}_1 \wedge \mathrm{a}_2 = \mathrm{b}_2 \wedge \mathrm{P}_1 \equiv \mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1' \wedge \mathrm{P}_2 \equiv \mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2' & \\
& \qquad\qquad \wedge (\mathtt{E}_1\left[\mathrm{x}[\mathrm{a}, \mathrm{b}] \mid \mathrm{P}_1'\right]; \mathtt{E}_2\left[\mathrm{x}[\mathrm{a}, \mathrm{b}] \mid \mathrm{P}_2'\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma \setminus \mathrm{a}, \mathrm{b}]\!] & \\
& \quad \wedge (\mathrm{a}_1 \in \mathrm{dom}(\Gamma) \wedge \mathrm{b}_1 \notin \mathrm{dom}(\Gamma)) \implies \big(\mathrm{a}_1 = \mathrm{a}_2 \wedge \mathrm{b}_2 \notin \mathrm{dom}(\Gamma) & \\
& \qquad \wedge \mathrm{P}_1 \equiv (\nu \mathrm{b}_1\mathrm{b}')(\mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1') \wedge \mathrm{P}_2 \equiv (\nu \mathrm{b}_2\mathrm{b}')(\mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2') & \\
& \qquad \wedge (\mathtt{E}_1\left[(\nu \mathrm{b}_1\mathrm{b}')(\mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1')\right]; \mathtt{E}_2\left[(\nu \mathrm{b}_2\mathrm{b}')(\mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2')\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma, \mathrm{b}' : \mathsf{B}[c] \setminus \mathrm{a}]\!]\big) & \\
& \quad \wedge (\mathrm{a}_1 \notin \mathrm{dom}(\Gamma) \wedge \mathrm{b}_1 \in \mathrm{dom}(\Gamma)) \implies \big(\mathrm{a}_2 \notin \mathrm{dom}(\Gamma) \wedge \mathrm{b}_1 = \mathrm{b}_2 & \\
& \qquad \wedge \mathrm{P}_1 \equiv (\nu \mathrm{a}_1\mathrm{a}')(\mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1') \wedge \mathrm{P}_2 \equiv (\nu \mathrm{a}_2\mathrm{a}')(\mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2') & \\
& \qquad \wedge (\mathtt{E}_1\left[(\nu \mathrm{a}_1\mathrm{a}')(\mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1')\right]; \mathtt{E}_2\left[(\nu \mathrm{a}_2\mathrm{a}')(\mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2')\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma, \mathrm{a}' : \mathsf{C}[c] \setminus \mathrm{b}]\!]\big) & \\
& \quad \wedge \mathrm{a}_1, \mathrm{b}_1 \notin \mathrm{dom}(\Gamma) \implies \big(\mathrm{a}_2, \mathrm{b}_2 \notin \mathrm{dom}(\Gamma) & \\
& \qquad \wedge \mathrm{P}_1 \equiv (\nu \mathrm{a}_1\mathrm{a}')(\nu \mathrm{b}_1\mathrm{b}')(\mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1') \wedge \mathrm{P}_2 \equiv (\nu \mathrm{a}_2\mathrm{a}')(\nu \mathrm{b}_2\mathrm{b}')(\mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2') & \\
& \qquad \wedge (\mathtt{E}_1\left[(\nu \mathrm{a}_1\mathrm{a}')(\nu \mathrm{b}_1\mathrm{b}')(\mathrm{x}[\mathrm{a}_1, \mathrm{b}_1] \mid \mathrm{P}_1')\right]; & \\
& \qquad\quad \mathtt{E}_2\left[(\nu \mathrm{a}_2\mathrm{a}')(\nu \mathrm{b}_2\mathrm{b}')(\mathrm{x}[\mathrm{a}_2, \mathrm{b}_2] \mid \mathrm{P}_2')\right]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma, \mathrm{a}' : \mathsf{A}[c], \mathrm{b}' : \mathsf{B}[c]]\!]\big) &
\end{array}
$$

Figure 5 Value interpretation, output cases ($1, \oplus, \otimes$).

of P_1' and P_2' to coincide. In particular, if P_1' does not produce any observable messages along a name in the interface due to a deadlock imposed by a secret, Part (8) guarantees that P_2' does not produce any observable messages on that name either.

We present our value interpretation, denoted $\mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma]\!]$, in Figs. 5 and 6. It relates pairs of context-process tuples $(\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2)$, given a secrecy lattice Ω, a secrecy level $\xi \in \mathrm{dom}(\Omega)$, an (observable) interface Γ, and a name $\mathrm{x} \in \mathrm{dom}(\Gamma)$. The relation is defined by cases on the type A assigned to x in Γ. If A is output-like ($1, \oplus, \otimes$; Fig. 5), the relation looks for a corresponding output on x in the processes to (observably) move across the interface into the contexts[2]; if A is input-like ($\bot, \&, \parr$; Fig. 6), the relation looks for a corresponding output on a name y connected by restriction to x in the contexts to (observably) move across the interface into the processes. We detail the representative cases where $\mathsf{A} \in \{\oplus, \&\}$.

When $\mathsf{A} = \oplus\{\mathrm{i} : \mathsf{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}$ (9), we first check well typedness as usual (10) (cf. Def. 4.5). We then assert that both P_1 and P_2 have ready a selection on x, both on the same label $\mathrm{j} \in \mathrm{I}$ (11).

[2] In the rest of the paper, we often write $\Gamma \setminus \mathrm{x}$ to denote Γ' given $\Gamma = \Gamma', \mathrm{x} : \mathsf{A}[c]$.

$$\bot \quad (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma, \mathrm{x} : \bot[c]]\!] \iff \big((\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma, \mathrm{x} : \bot[c])$$
$$\wedge \big(\mathtt{E}_1 \equiv (\nu \mathrm{yx})(\mathrm{y}[] \mid \mathtt{E}_1') \wedge \mathtt{E}_2 \equiv (\nu \mathrm{yx})(\mathrm{y}[] \mid \mathtt{E}_2')\big)$$
$$\implies (\mathtt{E}_1'\big[(\nu \mathrm{yx})(\mathrm{y}[] \mid \mathrm{P}_1)\big]; \mathtt{E}_2'\big[(\nu \mathrm{yx})(\mathrm{y}[] \mid \mathrm{P}_2)\big]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma]\!]\big)$$

$$\& \quad (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma, \mathrm{x} : \&\{\mathrm{i} : \mathrm{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}[c]]\!] \iff \tag{14}$$
$$(\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma, \mathrm{x} : \&\{\mathrm{i} : \mathrm{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}[c]) \tag{15}$$
$$\wedge \big(\exists \mathrm{j} \in \mathrm{I}.\ \mathtt{E}_1 \equiv (\nu \mathtt{bb}')(\nu \mathrm{yx})(\mathrm{y}[\mathrm{b}] \triangleleft \mathrm{j} \mid \mathtt{E}_1') \wedge \mathtt{E}_2 \equiv (\nu \mathtt{bb}')(\nu \mathrm{yx})(\mathrm{y}[\mathrm{b}] \triangleleft \mathrm{j} \mid \mathtt{E}_2')\big) \tag{16}$$
$$\implies ((\nu \mathtt{bb}')\mathtt{E}_1'\big[(\nu \mathrm{yx})(\mathrm{y}[\mathrm{b}] \triangleleft \mathrm{j} \mid \mathrm{P}_1)\big]; \tag{17}$$
$$(\nu \mathtt{bb}')\mathtt{E}_2'\big[(\nu \mathrm{yx})(\mathrm{y}[\mathrm{b}] \triangleleft \mathrm{j} \mid \mathrm{P}_2)\big]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma, \mathrm{b} : \mathrm{A}_\mathrm{j}[c]]\!]$$

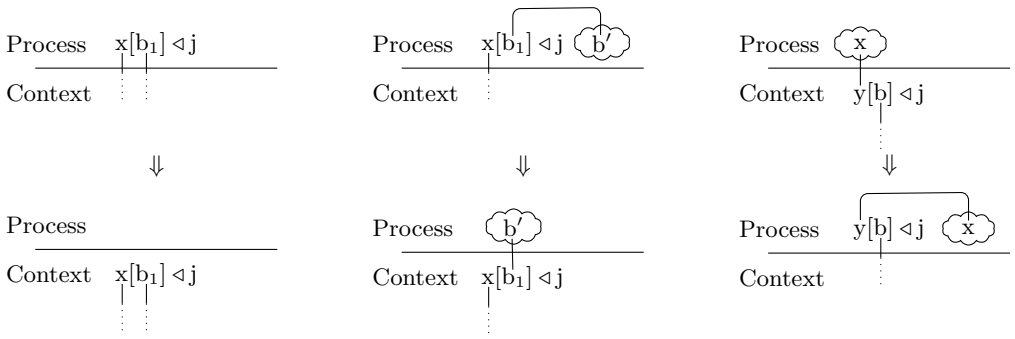
$$\parr \quad (\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{V}_{\mathrm{x}}^{\Omega;\xi}[\![\Gamma, \mathrm{x} : \mathrm{A} \parr \mathrm{B}[c]]\!] \iff$$
$$(\mathtt{E}_1, \mathrm{P}_1; \mathtt{E}_2, \mathrm{P}_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma, \mathrm{x} : \mathrm{A} \parr \mathrm{B}[c])$$
$$\wedge \big(\mathtt{E}_1 \equiv (\nu \mathtt{aa}')(\nu \mathtt{bb}')(\nu \mathrm{yx})(\mathrm{y}[\mathrm{a}, \mathrm{b}] \mid \mathtt{E}_1') \wedge \mathtt{E}_2 \equiv (\nu \mathtt{aa}')(\nu \mathtt{bb}')(\nu \mathrm{yx})(\mathrm{y}[\mathrm{a}, \mathrm{b}] \mid \mathtt{E}_2')\big)$$
$$\implies ((\nu \mathtt{aa}')(\nu \mathtt{bb}')\mathtt{E}_1'\big[(\nu \mathrm{yx})(\mathrm{y}[\mathrm{a}, \mathrm{b}] \mid \mathrm{P}_1)\big];$$
$$(\nu \mathtt{aa}')(\nu \mathtt{bb}')\mathtt{E}_2'\big[(\nu \mathrm{yx})(\mathrm{y}[\mathrm{a}, \mathrm{b}] \mid \mathrm{P}_2)\big]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma, \mathrm{a} : \mathrm{A}[c], \mathrm{b} : \mathrm{B}[c]]\!]$$

Figure 6 Value interpretation, input cases ($\bot, \&, \parr$).

We find that the selections carry continuations b_1 and b_2, respectively. Since we intend to move the selections across the interface into the contexts, we need to inspect where these b_i are bound: in the context or in the process.

- If b_1 appears in the interface (12), it is bound in $\mathtt{E}_1$. We then assert that b_1 and b_2 actually represent the same name, and thus that b_2 is bound in $\mathtt{E}_2$. Next, we use structural congruence (Def. 3.3) to separate the selections from the rest of the processes. The case ends with a call on the term interpretation, where the selections have been moved into the contexts. Note that here we remove b_1 ($= \mathrm{b}_2$) from the interface, as the processes have relinquished control over this name to their respective contexts: we no longer need to monitor behavior on b_1. Fig. 7a illustrates this case.
- If b_1 does not appear in the interface (13), it is bound in P_1. We first assert that b_2 also does not appear in the interface, and thus is bound in P_2. We then use structural congruence to identify the names to which each b_i is bound – since they are both bound, we conveniently apply alpha conversion and use b' in both cases – , and to separate the selections from the rest of the processes. Finally, we call on the term interpretation, where the selections along with the binders $(\nu \mathrm{b}_\mathrm{i}\mathrm{b}')$ are moved into the contexts. Here, we add b' to the interface, as it must be used in the remainder of the processes, and thus must be monitored. Fig. 7b shows this case.

When $\mathrm{A} = \&\{\mathrm{i} : \mathrm{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}$ (14), the processes are expecting a selection from the contexts. We again start with the usual well typedness check (15) (cf. Def. 4.5). The purpose of our relation is to compare runs of the same process in different contexts, and so we cannot make assertions about the readiness of the contexts to make the required selection, or that these are selections of the same label. We therefore proceed only under the condition that indeed the contexts are both ready to select the same label (16). This condition uses structural congruence to identify the names in the contexts to which x is connected, conveniently referred to as y in both contexts. It also identifies the continuations b of the selections and the names b' to which they are connected, as well as the remainder of the contexts. It then calls on the term evaluation (17), where the selections along with the restrictions binding x to y are moved into the processes. As such, x is no longer in the interface, but now the continuations of the selections are: we add b to the interface. Fig. 7c illustrates this case.



(a) Case $\oplus$, b_1 in interface. **(b)** Case $\oplus$, b_1 not in interface. **(c)** Case $\&$.

Figure 7 Illustrations of the value interpretation on selections: the selection is moved to/from the process, influencing name connections through the interface. Names in clouds represent parts of the process where the name is used.

Finally, we use our logical relation to define *equivalence up to observable messages*. We say two processes are equivalent up to secrecy level ξ if they agree on their observable interface and they are related by the logical relation when placed inside any two arbitrary evaluation contexts (cf. Def. 4.1). This ensures that, regardless of the context in which the processes run, they will behave the same with respect to the observable interface.

▶ **Definition 4.9** (Equivalence up to Observable Messages). *The relation*

$$(\Omega \vdash \mathrm{P}_1 \mathbin{@} d_1 :: \Gamma_1) \equiv_\xi (\Omega \vdash \mathrm{P}_2 \mathbin{@} d_2 :: \Gamma_2)$$

holds if and only if $\Gamma_1 \Downarrow \xi = \Gamma_2 \Downarrow \xi = \Gamma$, *and for every* $\mathrm{E}_1, \mathrm{E}_2$ *such that* $\Omega \vdash \mathrm{E}_1[\mathrm{P}_1] \mathbin{@} d_1 :: \emptyset$ *and* $\Omega \vdash \mathrm{E}_2[\mathrm{P}_2] \mathbin{@} d_2 :: \emptyset$, $(\mathrm{E}_1[\mathrm{P}_1]; \mathrm{E}_2[\mathrm{P}_2]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma]\!]$ *and* $(\mathrm{E}_2[\mathrm{P}_2]; \mathrm{E}_1[\mathrm{P}_1]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma]\!]$.

▶ **Example 4.10.** Consider again the secure variant of $\mathrm{Gov}_{\mathrm{A}}^{H}$ from Example 3.2. Anticipating noninterference, it is straightforward to check that the continuations of the initial branch are equivalent up to observable messages:

$$\begin{aligned}&(\vdash (\nu \mathrm{a}_{\mathrm{I}}^{1'} \mathrm{a}_{\mathrm{I}}^{1})(\mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{act} \mid \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[]) \mathbin{@} L :: \mathrm{a}_{\mathrm{I}} : \oplus\{\mathsf{act} : 1, \mathsf{wait} : 1\}[H], \mathrm{a}_{L}^{1} : \bot[L])\\ \equiv_L &(\vdash (\nu \mathrm{a}_{\mathrm{I}}^{1'} \mathrm{a}_{\mathrm{I}}^{1})(\mathrm{a}_{\mathrm{I}}[\mathrm{a}_{\mathrm{I}}^{1'}] \triangleleft \mathrm{wait} \mid \mathrm{a}_{L}^{1}(); \mathrm{a}_{\mathrm{I}}^{1}[]) \mathbin{@} L :: \mathrm{a}_{\mathrm{I}} : \oplus\{\mathsf{act} : 1, \mathsf{wait} : 1\}[H], \mathrm{a}_{L}^{1} : \bot[L])\end{aligned}$$

The crucial part is that the different selections on a_{I} are unobservable.

On the other hand, consider also the insecure variant of $\mathrm{Gov}_{\mathrm{A}}^{H}$ from Example 3.2. Even though their typing contexts are equal (and, hence, so are the projections onto L), the continuations of the initial branch are *not* equivalent up to observable messages:

$$\begin{aligned}&(\vdash (\nu \mathrm{a}_{L}^{1'} \mathrm{a}_{L}^{1})(\mathrm{a}_{L}[\mathrm{a}_{L}^{1'}] \triangleleft \mathrm{inf}_1 \mid \mathrm{a}_{\mathrm{I}}^{1}(); \mathrm{a}_{L}^{1}[]) :: \mathrm{a}_{L} : \oplus\{\mathsf{inf}_1 : 1, \mathsf{inf}_2 : 1\}[L], \mathrm{a}_{\mathrm{I}}^{1} : \bot[H])\\ \not\equiv_L &(\vdash (\nu \mathrm{a}_{L}^{1'} \mathrm{a}_{L}^{1})(\mathrm{a}_{L}[\mathrm{a}_{L}^{1'}] \triangleleft \mathrm{inf}_2 \mid \mathrm{a}_{\mathrm{I}}^{1}(); \mathrm{a}_{L}^{1}[]) :: \mathrm{a}_{L} : \oplus\{\mathsf{inf}_1 : 1, \mathsf{inf}_2 : 1\}[L], \mathrm{a}_{\mathrm{I}}^{1} : \bot[H])\end{aligned}$$

Here, the different selections on a_L *are* observable.

5 Deadlock-Sensitive Noninterference (DSNI)

Our main result is that the observable behavior (up to a given secrecy level ξ) of any well-typed process is the same when placed in different contexts. We formalize this using our logical relation (Def. 4.9):

▶ **Theorem 5.1** (DSNI). *For all secrecy lattices Ω, secrecy levels $\xi \in \mathrm{dom}(\Omega)$ and processes $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma$, we have $(\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma) \equiv_\xi (\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma)$.*

▶ **Example 5.2.** Following up on Example 4.10, we can conclude that DSNI holds for the secure variant of $\mathrm{Gov}_{\mathrm{A}}^{H}$, but not for the insecure variant.

To prove this main result, we prove a more general result (the *fundamental theorem*; Thm. 5.7) that relates two processes through Def. 4.9 given that they are *observably equivalent*. We first define precisely what we mean with observable equivalence before presenting and proving our fundamental theorem in Sec. 5.2.

5.1 Observable Equivalence

Towards defining observable equivalence, we want to identify the nodes (cf. Def. 4.8) of processes that can contribute to messages on observable names in the interface, referred to as *relevant nodes*. Nodes with running secrecy $\not\sqsubseteq \xi$ obviously cannot influence observable interface names. However, nodes with running secrecy $\sqsubseteq \xi$ are not necessarily capable of influencing observable interface names either. In particular, two types of nodes with running secrecy $\sqsubseteq \xi$ cannot influence the observable interface:

- Nodes that input on unobservable names increase their running secrecy after the input, such that they no longer influence observable interface names.
- Nodes that output on unobservable names can only influence nodes that input on unobservable names (and thus cannot influence observable interface names indirectly via the receiving node).

The following notion of *quasi-running secrecy* anticipates these scenarios by assigning a secrecy level to a process based on the influence of its foremost prefix corresponding to the subsequent input/output. It is defined as the join of the current running secrecy of the process and the secrecy level of the name on which the next input/output occurs. If either of the two levels is unobservable, the quasi-running secrecy will be unobservable. In such cases, we know that the foremost prefix of the process cannot influence the observable interface.

▶ **Definition 5.3** (Quasi-running Secrecy). *Given a node typed $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma$, we define the* quasi-running secrecy *of* P*, denoted $\mathit{quasi}(\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma)$ as follows:*

$$\mathit{quasi}(\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma) := \begin{cases} d \sqcup c & \text{if } \mathrm{P} = \mathrm{x}[] \text{ and } \mathrm{x} : \mathsf{1}[c] \in \Gamma \\ d \sqcup c & \text{if } \mathrm{P} = \mathrm{x}(); \mathrm{P}' \text{ and } \mathrm{x} : \bot[c] \in \Gamma \\ d \sqcup c & \text{if } \mathrm{P} = \mathrm{x}[\mathrm{b}] \triangleleft \mathrm{j} \text{ and } \mathrm{x} : \oplus\{\mathrm{i} : \mathsf{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}[c] \in \Gamma \\ d \sqcup c & \text{if } \mathrm{P} = \mathrm{x}(\mathrm{z}) \triangleright \{\mathrm{i} : \mathrm{P}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}} \text{ and } \mathrm{x} : \&\{\mathrm{i} : \mathsf{A}_\mathrm{i}\}_{\mathrm{i}\in\mathrm{I}}[c] \in \Gamma \\ d \sqcup c & \text{if } \mathrm{P} = \mathrm{x}[\mathrm{a}, \mathrm{b}] \text{ and } \mathrm{x} : \mathsf{A} \otimes \mathsf{B}[c] \in \Gamma \\ d \sqcup c & \text{if } \mathrm{P} = \mathrm{x}(\mathrm{y}, \mathrm{z}); \mathrm{P}' \text{ and } \mathrm{x} : \mathsf{A} \parr \mathsf{B}[c] \in \Gamma \end{cases}$$

To compute which nodes of a process are relevant, we start with nodes that have connections to the interface (through free names). We then look at nodes that are connected to these relevant nodes through restrictions. However, not all connections imply a possible influence on the observable interface. Consider a node $\mathrm{x}[\mathrm{a}, \mathrm{b}]$ that is connected to a relevant node on a: the node does not define behavior on a but merely outputs the name, and so a cannot influence the observable interface through this name. For example, in $(\nu \mathrm{xy})(\nu \mathrm{au})(\mathrm{x}[\mathrm{a}, \mathrm{b}] \mid \mathrm{y}(\mathrm{z}, \mathrm{w}); \mathrm{z}(); \ldots \mid \mathrm{u}[])$, the name a is not used for communication until it has been received on y; hence, the close on u is not considered relevant even if the send on x were relevant. We make this precise by defining *free communication names*: free names that are used as the subjects of unblocked prefixes.

▶ **Definition 5.4** (Free Communication Names). *We define the* free communication names *of* P*, denoted* $\mathrm{fcn}(\mathrm{P})$*, as follows:*

$$\begin{aligned}
\mathrm{fcn}(0) &:= \emptyset \\
\mathrm{fcn}(\mathrm{P} \mid \mathrm{Q}) &:= \mathrm{fcn}(\mathrm{P}) \cup \mathrm{fcn}(\mathrm{Q}) & \mathrm{fcn}((\nu \mathrm{xy})\mathrm{P}) &:= \mathrm{fcn}(\mathrm{P}) \setminus \{\mathrm{x}, \mathrm{y}\} \\
\mathrm{fcn}(\mathrm{x}[]) &:= \{\mathrm{x}\} & \mathrm{fcn}(\mathrm{x}(); \mathrm{P}) &:= \{\mathrm{x}\} \cup \mathrm{fcn}(\mathrm{P}) \\
\mathrm{fcn}(\mathrm{x}[\mathrm{a}, \mathrm{b}]) &:= \{\mathrm{x}\} & \mathrm{fcn}(\mathrm{x}(\mathrm{y}, \mathrm{z}); \mathrm{P}) &:= \{\mathrm{x}\} \cup \mathrm{fcn}(\mathrm{P}) \setminus \{\mathrm{y}, \mathrm{z}\} \\
\mathrm{fcn}(\mathrm{x}[\mathrm{b}] \triangleleft \mathrm{j}) &:= \{\mathrm{x}\} & \mathrm{fcn}(\mathrm{x}(\mathrm{z}) \triangleright \{\mathrm{i} : \mathrm{P}_\mathrm{i}\}_{\mathrm{i} \in \mathrm{I}}) &:= \{\mathrm{x}\} \cup \textstyle\bigcup_{\mathrm{i} \in \mathrm{I}} \mathrm{fcn}(\mathrm{P}_\mathrm{i}) \setminus \{\mathrm{z}\}
\end{aligned}$$

We now have all the ingredients to determine the relevant nodes of a process. We define the set of relevant nodes of a process inductively by following chains of nodes connected through restriction (of which there are finitely many). We start with nodes connected to the interface directly, and add them if their quasi-running secrecy is $\sqsubseteq \xi$. We then keep adding nodes that are connected to already relevant nodes on observable channels (names with secrecy level $\sqsubseteq \xi$) with quasi-running secrecy $\sqsubseteq \xi$.

▶ **Definition 5.5** (Relevant Nodes and Binders, and Relevant Form). *Suppose given a process in normal form* P *typed* $\Omega \vdash \mathrm{P} \mathbin{@} d :: \Gamma$*. Suppose every node* $\mathrm{Q} \in \mathrm{nodes}(\mathrm{P})$ *is typed* $\Omega \vdash \mathrm{Q} \mathbin{@} d_\mathrm{Q} :: \Gamma_\mathrm{Q}$*. Given a secrecy level* $\xi \in \mathrm{dom}(\Omega)$*, we define the set of* relevant nodes *of* P*, denoted* $\mathrm{N}(\mathrm{P})$*, by induction on the size of* $\mathrm{binders}(\mathrm{P})$ *as follows (*$\mathrm{N}(\mathrm{P}) := \mathrm{N}_{|\mathrm{binders}(\mathrm{P})|}(\mathrm{P})$*):*

$$\begin{aligned}
\mathrm{N}_0(\mathrm{P}) &:= \{\mathrm{Q} \in \mathrm{nodes}(\mathrm{P}) \mid \exists \mathrm{z} \in \mathrm{fcn}(\mathrm{Q}).\ \mathrm{z} \in \mathrm{dom}(\Gamma \Downarrow \xi) \wedge \mathit{quasi}(\Omega \vdash \mathrm{Q} \mathbin{@} d_\mathrm{Q} :: \Gamma_\mathrm{Q}) \sqsubseteq \xi\} \\
\mathrm{N}_{n+1}(\mathrm{P}) &:= \mathrm{N}_n(\mathrm{P}) \cup \left\{ \mathrm{Q} \in \mathrm{nodes}(\mathrm{P}) \;\middle|\; \begin{array}{l} \exists \mathrm{z} \in \mathrm{fcn}(\mathrm{Q}).\ \exists \mathrm{Q}' \in \mathrm{N}_n(\mathrm{P}).\ \exists \mathrm{w} : \mathsf{A}_\mathrm{w}[c] \in \Gamma_{\mathrm{Q}'}. \\ \quad (\Omega \Vdash c \sqsubseteq \xi \wedge \{\mathrm{z}, \mathrm{w}\} \in \mathrm{binders}(\mathrm{P})) \\ \wedge\ \mathit{quasi}(\Omega \vdash \mathrm{Q} \mathbin{@} d_\mathrm{Q} :: \Gamma_\mathrm{Q}) \sqsubseteq \xi \end{array} \right\} \\
&\quad \forall 0 \leq n < |\mathrm{binders}(\mathrm{P})|
\end{aligned}$$

We also define the set of relevant binders *of* P*, denoted* $\mathrm{B}(\mathrm{P})$*, as the subset of* $\mathrm{binders}(\mathrm{P})$ *used in the inductive step of the definition of* $\mathrm{N}(\mathrm{P})$*. We then define the* relevant form *of a process in normal form* P*, denoted* $\mathrm{P} \Downarrow \xi$*, as* $(\nu \mathrm{xy})_{\{\mathrm{x},\mathrm{y}\} \in \mathrm{B}(\mathrm{P})} \prod_{\mathrm{Q} \in \mathrm{N}(\mathrm{P})} \mathrm{Q}$*.*

Processes are then observably equivalent if their relevant nodes and relevant binders are indistinguishable (up to structural congruence).

▶ **Definition 5.6** (Observable Equivalence). *We say that two processes* P *and* P' *are* observably equivalent*, denoted* $\mathrm{P} \equiv_\xi \mathrm{P}'$*, if and only if there are normal forms* Q, Q' *of* P, P' *respectively such that* $\mathrm{Q} \Downarrow \xi \equiv \mathrm{Q}' \Downarrow \xi$*.*

5.2 The Fundamental Theorem

We now state and prove our fundamental theorem, from which DSNI (Thm. 5.1) follows.

▶ **Theorem 5.7** (Fundamental Theorem). *For all secrecy lattices* Ω*, secrecy levels* $\xi \in \mathrm{dom}(\Omega)$ *and processes* $\Omega \vdash \mathrm{P}_1 \mathbin{@} d_1 :: \Gamma_1$ *and* $\Omega \vdash \mathrm{P}_2 \mathbin{@} d_2 :: \Gamma_2$ *with* $\mathrm{P}_1 \equiv_\xi \mathrm{P}_2$ *and* $\Gamma_1 \Downarrow \xi = \Gamma_2 \Downarrow \xi$*, we have* $(\Omega \vdash \mathrm{P}_1 \mathbin{@} d_1 :: \Gamma_1) \equiv_\xi (\Omega \vdash \mathrm{P}_2 \mathbin{@} d_2 :: \Gamma_2)$*.*

We first give several auxiliary results and definitions, before proving Thm. 5.7 on Page 23:

- Lem. 5.8 splits a process that reduces into an evaluation context (Def. 4.1) containing the source of the reduction originating from one of the reduction axioms in Fig. 2 (bottom).
- Lem. 5.9 splits unobservable reduction (Def. 4.7) into one of three cases: reduction internal in the context, reduction internal in the process, and communication between context and process on unobservable names.

- Lem. 5.10 asserts that two observably equivalent (Def. 5.6) processes can "catch up" on each other's unobservable reductions (Def. 4.7). That is, if one process reduces unobservably, then the other process can do zero or one unobservable reductions such that the resulting processes are again observably equivalent.
- Def. 5.11 defines a *weight* on types and typing context, which we use for induction in the proof of Thm. 5.7 on Page 23.

Lems. 5.8 and 5.9 are proven in the extended paper.

▶ **Lemma 5.8.** *Suppose given a process typed* $\Omega \vdash P \mathbin{@} d :: \Gamma$. *If* $P \longrightarrow P'$, *then there exists an* E *for which either of the following holds:*

1. $P \equiv E[(\nu xy)(x[] \mid y(); Q)]$ *and* $P' \equiv E[Q]$;
2. $P \equiv E[(\nu xy)(x[a, b] \mid y(z, w); Q)]$ *and* $P' \equiv E[Q\{a/z, b/w\}]$;
3. $P \equiv E[(\nu xy)(x[b] \lhd j \mid y(w) \rhd \{i : Q_i\}_{i \in I})]$ *for* $j \in I$ *and* $P' \equiv E[Q_j\{b/w\}]$.

▶ **Lemma 5.9.** *Suppose* $(E, P) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$ *and* $E, P \longrightarrow_{\Omega;\xi;\Gamma} E', P'$. *Then* $E \longrightarrow E'$ *and* $P = P'$, *or* $P \longrightarrow P'$ *and* $E = E'$, *or Lem. 5.8 applies, on names not in* Γ.

▶ **Lemma 5.10** (Catch Up). *Suppose* $(E_1, P_1; E_2, P_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$ *such that* $P_1 \equiv_\xi P_2$. *If* $E_1, P_1 \longrightarrow_{\Omega;\xi;\Gamma} E_1', P_1'$, *then there exists* P_2' *such that* $E_2, P_2 \longrightarrow^{?}_{\Omega;\xi;\Gamma} E_2, P_2'$ *and* $P_1' \equiv_\xi P_2'$.

Proof. For a smoother proof, we consider a normal form Q_1 of P_1, and obtain from Q_1 a normal form Q_2 of P_2 such that $Q_1 \Downarrow \xi = Q_2 \Downarrow \xi$. By Def. 4.8, the thesis follows by proving the thesis for these Q_1, Q_2.

By Lem. 5.9, we can distinguish three cases from which $E_1, Q_1 \longrightarrow_{\Omega;\xi;\Gamma} E_1', Q_1'$ follows.

- **(Internal in context:** $E_1 \longrightarrow E_1'$ **and** $Q_1 = Q_1'$**)** The thesis holds directly with $Q_2' := Q_2$.
- **(Internal in process:** $Q_1 \longrightarrow Q_1'$ **and** $E_1 = E_1'$**)** By Lem. 5.8, Q_1's reduction is due to one of three possible synchronizations inside some evaluation context. Note that Lem. 5.8 may give us processes that are alpha variant to Q_1 and Q_1'; in the following we implicitly apply further alpha renaming to match the names in Q_1 and Q_1'. For space considerations, we sketch only the **(Close-Wait)** case; the other two cases are analogous. Full details are in the extended paper.

  We have $Q_1 \equiv F_1[(\nu xy)(x[] \mid y(); R)] \longrightarrow F_1[R] \equiv Q_1'$. The analysis depends on whether the close on x is a relevant node of Q_1 or not.

  If not, we derive that the wait on y is also not relevant. It follows by well typedness that the continuation R will neither add relevant nodes nor influence relevancy of other nodes, so $Q_1 \Downarrow \xi = Q_1' \Downarrow \xi$ and the thesis follows with $Q_2' := Q_2$.

  If the close is indeed a relevant node of Q_1, we derive that the wait on y and the binder between x and y are also relevant. By assumption, they are then also relevant in Q_2, so we can derive a similar reduction to Q_2'.

  It remains to show that $Q_1' \Downarrow \xi \equiv Q_2' \Downarrow \xi$, which boils down to showing that these processes have coinciding sets of relevant nodes and binders. Both directions of these set inclusions are analogous, so we focus on one: from Q_1' to Q_2'. The analysis is by induction on the construction of the sets of relevant nodes and binders. In each case, we consider the appearance of the node: in R or in F_1. In both cases, a thorough analysis of how the node was included as a relevant node – through a path of relevant binders and nodes that were added before – reveals an analogous relevant node in Q_2'.
- **(Communication between context and process on names not in** Γ**)** By definition, the secrecy levels of the involved names are incomparable to ξ. Therefore, none of the nodes involved are relevant or influence relevancy of any other nodes, so $Q_1 \Downarrow \xi = Q_1' \Downarrow \xi$ and the thesis holds with $Q_2' := Q_2$. ◀

▶ **Definition 5.11** (Weight). *The weight of a type* A*, denoted* $\varpi(A)$*, is defined as follows:*

$$\varpi(1) := 1 \qquad \varpi(A \otimes B) := \varpi(A) + \varpi(B) + 1 \qquad \varpi(\oplus\{i : A_i\}_{i\in I}) := \max_{i\in I}(\varpi(A_i)) + 1$$
$$\varpi(\bot) := 1 \qquad \varpi(A \parr B) := \varpi(A) + \varpi(B) + 1 \qquad \varpi(\&\{i : A_i\}_{i\in I}) := \max_{i\in I}(\varpi(A_i)) + 1$$

The weight of a typing context $\varpi(\Gamma)$ *is the sum of the weights of its types.*

▶ **Theorem 5.7** (Fundamental Theorem). *For all secrecy lattices* Ω*, secrecy levels* $\xi \in \mathrm{dom}(\Omega)$ *and processes* $\Omega \vdash P_1 \mathbin{@} d_1 :: \Gamma_1$ *and* $\Omega \vdash P_2 \mathbin{@} d_2 :: \Gamma_2$ *with* $P_1 \equiv_\xi P_2$ *and* $\Gamma_1 \Downarrow \xi = \Gamma_2 \Downarrow \xi$*, we have* $(\Omega \vdash P_1 \mathbin{@} d_1 :: \Gamma_1) \equiv_\xi (\Omega \vdash P_2 \mathbin{@} d_2 :: \Gamma_2)$.

Proof. Let $\Gamma := \Gamma_1 \Downarrow \xi = \Gamma_2 \Downarrow \xi$. Take any E_1, E_2 such that $\Omega \vdash E_1[P_1] \mathbin{@} d_1' :: \emptyset$ and $\Omega \vdash E_2[P_2] \mathbin{@} d_2' :: \emptyset$. We need to show that $(E_1[P_1]; E_2[P_2]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma]\!]$, which we do by induction on $\varpi(\Gamma)$.

The first condition is that $(E_1, P_1; E_2, P_2) \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$; this holds by assumption.

Next, take any E_1', P_1' such that $E_1, P_1 \longrightarrow^*_{\Omega;\xi;\Gamma} E_1', P_1' \not\longrightarrow_{\Omega;\xi;\Gamma}$. A straightforward induction on the length of these unobservable reductions shows that, by Def. 4.7 and Lem. 5.10, there are E_2', P_2' such that $E_2, P_2 \longrightarrow^*_{\Omega;\xi;\Gamma} E_2', P_2' \not\longrightarrow_{\Omega;\xi;\Gamma}$, $(E_1', P_1'; E_2', P_2') \in \mathsf{Net}^{\Omega;\xi}(\Gamma)$, and $P_1' \equiv_\xi P_2'$.

Now, we need to show that, for every $x \in \big(\mathrm{ain}(E_1', P_1') \cup \mathrm{ain}(E_2', P_2')\big) \cap \mathrm{dom}(\Gamma)$,

$$(E_1', P_1'; E_2', P_2') \in \mathsf{V}_x^{\Omega;\xi}[\![\Gamma]\!].$$

Take any such x. Either $x \in \mathrm{ain}(E_1', P_1')$ or $x \in \mathrm{ain}(E_2', P_2')$; w.l.o.g., assume the former. The rest of the analysis depends on the type of x in Γ.

First, we discuss the output-like cases $(1, \oplus, \otimes)$. In each case, by well typedness, x is the subject of an output-like prefix in P_1'. Since $x \in \mathrm{ain}(E_1', P_1')$, this prefix is unguarded. Since $x \in \mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma_1 \Downarrow \xi)$, the node in which the prefix appears is relevant in P_1'. Therefore, since $P_1' \equiv_\xi P_2'$, there is also a relevant node in P_2' where this prefix appears unguarded.

For space considerations, we only detail the case where x has type $\oplus\{i : A_i\}[c]$; the other cases are discussed in the extended paper. There exists $j \in I$ such that $x[b_1] \triangleleft j \in \mathrm{nodes}(P_1')$ and $x[b_2] \triangleleft j \in \mathrm{nodes}(P_1')$. The analysis depends on whether $b_1 \in \mathrm{dom}(\Gamma)$ or not.

- $(b_1 \in \mathrm{dom}(\Gamma))$ By well typedness, $b_1 \in \mathrm{fn}(P_1') \cap \mathrm{fn}(P_2')$. Since $P_1' \equiv_\xi P_2'$, then $b_1 = b_2$. Hence, $P_1' \equiv x[b_1] \triangleleft j \mid P_1''$ and $P_2' \equiv x[b_2] \triangleleft j \mid P_2''$. Similar to the case above, and since $\varpi(\Gamma \setminus x) < \varpi(\Gamma)$, it follows from the IH that $(E_1'[x[b_1] \triangleleft j \mid P_1'']; E_2'[x[b_2] \triangleleft j \mid P_2'']) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma \setminus x]\!]$. This proves that $(E_1', P_1'; E_2', P_2') \in \mathsf{V}_x^{\Omega;\xi}[\![\Gamma]\!]$.
- $(b_1 \notin \mathrm{dom}(\Gamma))$ By well typedness, $P_1' \equiv (\nu b_1 b')(x[b_1] \triangleleft j \mid P_1'')$. The selection on x is a relevant node of P_1'. Since $P_1' \equiv_\xi P_2'$, it is also a relevant node of P_2'. Moreover, $b_2 \notin \mathrm{fn}(P_2')$: otherwise, $b_2 = b_1$, and then $b_1 \in \mathrm{fn}(P_1')$. Hence, $P_2' \equiv (\nu b_2 b')(x[b_2] \triangleleft j \mid P_2'')$. Clearly, $\Omega \vdash P_1'' \mathbin{@} d_1'' :: \Gamma_1 \setminus x, b'$ and $\Omega \vdash P_2'' \mathbin{@} d_2'' :: \Gamma_2 \setminus x, b'$, and $\Omega \vdash E_1'[(\nu b_1 b')(x[b_1] \triangleleft j \mid P_1'')] \mathbin{@} d_1''' :: \emptyset$ and $\Omega \vdash E_2'[(\nu b_2 b')(x[b_2] \triangleleft j \mid P_2'')] \mathbin{@} d_2''' :: \emptyset$. Again, since $P_1' \equiv_\xi P_2'$, the chain of nodes and binders that are relevant in P_1' through the binder $(\nu b_1 b')$ has an equivalent such chain in P_2' through $(\nu b_2 b')$ and the selection on b_2. Hence, the effect on relevant nodes and binders by removing the binder and the selection on x is the same on P_1'' as it is on P_2'': $P_1'' \equiv_\xi P_2''$. Clearly, $\Gamma_1 \setminus x, b' \Downarrow \xi = \Gamma_2 \setminus x, b' = \Gamma \setminus x, b'$. Also, $\varpi(A_j) < \varpi(\oplus\{A_i\}_{i\in I})$, so $\varpi(\Gamma \setminus x, b') < \varpi(\Gamma)$. It then follows from the IH that $(E_1'[(\nu b_1 b')(x[b_1] \triangleleft j \mid P_1'')]; E_2'[(\nu b_2 b')(x[b_2] \triangleleft j \mid P_2'')]) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma \setminus x, b']\!]$. This proves that $(E_1', P_1'; E_2', P_2') \in \mathsf{V}_x^{\Omega;\xi}[\![\Gamma]\!]$.

Next, we discuss the negative cases ($\bot, \&, ⅋$). In each case, by well typedness, x is the subject of an input-like prefix in P_1'. The context E_1' binds x to some y by restriction, and E_1' contains a complementary output-like prefix on y. Following similar reasoning, the same holds for E_2'. Since $x \in ain(E_1', P_1')$, this output-like prefix appears unguarded in E_1'. To prove the thesis, we assume that this prefix also appears unguarded in E_2'.

For space considerations, we only detail the case where x has type $\bot[c]$; the other cases require additional care in handling continuation endpoints and are discussed in the extended paper. We have $E_1' \equiv (\nu yx)(y[] \mid E_1'')$ and $E_2' \equiv (\nu yx)(y[] \mid E_2'')$. Let $P_1'' := (\nu yx)(y[] \mid P_1')$ and $P_2'' := (\nu yx)(y[] \mid P_2')$. Clearly, $\Omega \vdash P_1'' \mathbin{@} d_1'' :: \Gamma_1 \setminus x$ and $\Omega \vdash P_2'' \mathbin{@} d_2'' :: \Gamma_2 \setminus x$, and $\Omega \vdash E_1''[P_1''] \mathbin{@} d_1''' :: \emptyset$ and $\Omega \vdash E_2''[P_2''] \mathbin{@} d_2''' :: \emptyset$.

Let Q_1, Q_2 denote the nodes of P_1', P_2', respectively, in which x appears. To prove that $P_1'' \equiv_\xi P_2''$, it suffices to show that Q_1 and any related binders are relevant in P_1'' if and only if Q_2 and any related binders are relevant in P_2''; any connected nodes/binders follow similar reasoning. We detail only the left-to-right direction; the other direction is analogous. Suppose Q_1 is relevant in P_1''. Then $quasi(Q_1) \sqsubseteq \xi$, and thus Q_1 is also relevant in P_1' through x in the interface. Then also Q_2 is relevant in P_2', where $Q_1 \equiv Q_2$ and $quasi(Q_2) \sqsubseteq \xi$. The analysis depends on how Q_1 is relevant in P_1'': (i) through the interface, or (ii) through a restriction with another relevant node. In case (i), it follows straightforwardly that Q_2 is also relevant in P_2'. In case (ii), the connected node is also relevant in P_1', and hence there is a related node that is also relevant in P_2'. Since the two processes agree on observable channels, the channel responsible for including Q_1 as a relevant node of P_1'' is also bound in P_2''. Then we can conclude that Q_2 is a relevant node of P_2''.

Since $\varpi(\Gamma \setminus x) < \varpi(\Gamma)$, it then follows from the IH that $(E_1''[P_1'']; E_2''[P_2'']) \in \mathsf{E}^{\Omega;\xi}[\![\Gamma \setminus x]\!]$. This proves that $(E_1', P_1'; E_2', P_2') \in \mathsf{V}_x^{\Omega;\xi}[\![\Gamma]\!]$.

Finally, we show that $aon(P_1') \cap dom(\Gamma) = aon(P_2') \cap dom(\Gamma)$. To prove this set equality, we take any $x \in aon(P_1') \cap dom(\Gamma)$ and prove that $x \in aon(P_2') \cap dom(\Gamma)$; the other direction is analogous. Clearly, x is the subject of an output-like prefix in P_1'. Since $x \in dom(\Gamma)$, this output-like prefix must appear unguarded in a node in P_1'. If the quasi-running secrecy of this node is observable, this node is relevant in P_1'. Since $P_1' \equiv_\xi P_2'$, P_2' must also have a relevant node in which the output-like prefix appears unguarded. Otherwise, the node is not relevant in P_1', and hence the node in which the output-like prefix appears in P_2' is also not relevant in P_2'. Hence, $x \in aon(P_2') \cap dom(\Gamma)$. ◀

DSNI and deadlock freedom. As mentioned in Sec. 3.1, our process language is based on the finite fragment of APCP with priority mechanisms removed. By enriching our process language with APCP's priority mechanisms, we restrict well typedness to deadlock-free processes. As such, our results remain relevant if we only consider deadlock-free processes.

6 Related Work

Logical relations for session types. Existing logical relations for session types are primarily unary, focusing on proving termination [52, 53, 21]. Binary logical relations have been contributed for proving parametricity [8] and noninterference [20, 5]. All of these logical relations are developed for intuitionistic linear session types, where process networks form trees and, as a result, neither permit cyclic networks nor deadlocks. Whereas our logical relation has its foundations in linear session types, it differs in that it is based on classical linear logic and allows for cycles and deadlocks.

Our work is most similar to prior work by Derakhshan et al. [20] and Balzer et al. [5] on a binary logical relation, in which the authors develop a flow-sensitive IFC type system and use the logical relation to prove noninterference. Our IFC type system is also flow sensitive, but it is designed for an adaptation of APCP (the background of which we discuss separately) that allows for cyclic process networks and deadlocks. Our logical relation resembles the one by the authors in that it employs an interface of names along which observations can be made. In contrast to Derakhshan et al. [20] and Balzer et al. [5], our interface is a set of names with types, rather than a sequent that singles out the providing name from the names being used. The distinction becomes necessary in an intuitionistic linear logic setting, whereas our work is grounded in classical linear logic. The contrast between the intuitionistic and classical setting manifests itself in other aspects of our development, too. For example, in prior intuitionistic IFC session type systems [20, 5], the offering channel always caps the secrecy of the channels in the context and the running secrecy. In our setting, however, the running secrecy of a process and the secrecy levels of its context are not necessarily related. In particular, waiting for a channel to close may increase the running secrecy of a process. Once the channel is closed, it disappears from the context, leaving the running secrecy entirely unrelated to the secrecies of the remaining channels.

Like our language, Derakhshan et al. [20]'s language lacks any recursion construct. On the other hand, the possibility of deadlocks introduces a side channel similar to non-termination, which is present in the work by Balzer et al. [5] that supports general recursive session types and thus possibly looping processes. Here, we decided to consider non-recursive processes to focus on side channels through deadlocks only and not through non-termination. In future work we would like to consider scaling our work to support general recursive session types as well. We envision employing an observation index similar to Balzer et al. [5] to stratify the logical relation in the number of observable messages exchanged over the interface. The co-presence of both non-termination and deadlocks will need careful consideration.

Our idea of an observable interface is reminiscent of the free channels with visible communications in prior work by Atkey [4]. Atkey establishes observational equivalence for Wadler's Classical Processes (CP) by defining a denotational semantics for CP and a logical-relations argument. However, the logical relation in Atkey's work does not relate two processes of a certain type but rather identifies the possible observations for each type in terms of the input/output behavior of its connectives.

Cyclic process networks. Traditionally, (typed) π-calculi permit cyclic process networks, and as such do not guarantee deadlock freedom. However, since the discovery of Curry-Howard correspondences between linear logic and session types [9, 67], the majority of works on session types restrict their network shapes to trees, such that deadlock freedom is guaranteed. The line of work including APCP (to which our process language is highly related) [42, 51, 19, 29, 30] considers restrictions to session type systems that allow cyclic process networks without deadlocks.

IFC type systems for multiparty session types. Capecchi et al. [13, 11] explore secure information flow with controlled forms of declassification for multiparty sessions and prove a noninterference result via a bisimulation. Our work differs in being flow sensitive and using a binary rather than multiparty session type paradigm. Our use of a logical relation to show noninterference, and our foundations in linear logic also set us apart. Follow-up works by Castellani et al. [14, 12] also study run-time monitoring techniques to ensure secure information flow control in multiparty sessions.

IFC type systems for process calculi. Several approaches have been explored for designing IFC type systems that prevent the leakage of information through message passing in process calculi [34, 35, 15, 16, 17, 36, 18, 26, 25, 41, 68, 55]. Some of these approaches include associating a security label with types or channels [36], associating a security label with actions [35], associating read and write policies with channels [26, 25], and associating a security label with processes and capabilities with expressions [15]. Our approach differs from previous work in having a dynamic running secrecy that makes our system flow sensitive, using session types instead of process calculi, and the design of our novel logical relations for establishing noninterference.

Logical relations for stateful languages. Kripke logical relations have been used to reason about stateful programs [54]. The relation is indexed by a possible world that serves as a semantic model for the heap. It establishes an invariant on the heap and ensures that the invariant is preserved for all future worlds. When combined with step indexing [3, 1], Kripke logical relations can address circularity that arises in higher-order stores [2, 22, 23]. Our logical relation is similar to Kripke logical relations in being developed for a stateful language; names, like locations, are subject to concurrent mutation. However, our session types are rooted in linear logic and thus internalize Kripke's logical worlds into the type system.

7 Conclusions

We have presented a new session type system with information flow control (IFC) for an asynchronous π-calculus, and a notion of noninterference by means of a logical relation between typed processes. Our development flexibly supports realistic cyclic process networks that may deadlock. As such, our main result is that IFC well typedness implies deadlock-sensitive noninterference (DSNI).

In future work, we plan to study the interplay between IFC / DSNI and several interesting features of message-passing concurrency, such as recursion and non-determinism. As commented on in the previous section, we expect the notion of an observation index [5] to be applicable to circular process networks with recursive processes, although the interplay between potential leakage through non-termination and deadlocks will need careful consideration. Support of non-determinism may be more challenging, especially when combining recursion with non-deterministic choice, which without further restriction permits loop guards at mixed confidentiality level. We will consider focusing on more curtailed but logically motivated notions of non-determinism, such as coexponentials [56], $\mathrm{HCP}^{-}_{\mathrm{ND}}$ [43], and linear non-determinism [28]. We are also interested in exploring "name-sensitive" noninterference: the choice of names in outputs is a possible source of information leakage outside the scope of this paper.

References

1 Amal Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *15th European Symposium on Programming (ESOP)*, volume 3924 of *Lecture Notes in Computer Science*, pages 69–83. Springer, 2006. doi:10.1007/11693024_6.

2 Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 340–353. ACM, 2009. doi:10.1145/1480881.1480925.

3 Andrew W. Appel and David A. McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(5):657–683, 2001. doi:10.1145/504709.504712.

4 Robert Atkey. Observed communication semantics for classical processes. In *Programming Languages and Systems: 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings 26*, pages 56–82. Springer, 2017.

5 Stephanie Balzer, Farzaneh Derakhshan, Robert Harper, and Yue Yao. Logical relations for session-typed concurrency. *CoRR*, abs/2309.00192, 2023. doi:10.48550/arXiv.2309.00192.

6 Stephanie Balzer and Frank Pfenning. Manifest sharing with session types. *Proceedings of the ACM on Programming Languages*, 1(ICFP):37:1–37:29, 2017. doi:10.1145/3110281.

7 Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In *28th European Symposium on Programming (ESOP)*, volume 11423 of *Lecture Notes in Computer Science*, pages 611–639. Springer, 2019. doi:10.1007/978-3-030-17184-1_22.

8 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *22nd European Symposium on Programming (ESOP)*, pages 330–349, 2013. doi:10.1007/978-3-642-37036-6_19.

9 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *21th International Conference onf Concurrency Theory (CONCUR)*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.

10 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.

11 Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Typing access control and secure information flow in sessions. *Information and Computation*, 238:68–105, 2014. doi:10.1016/j.ic.2014.07.005.

12 Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Information flow safety in multiparty sessions. *Mathematical Structures in Computer Science*, 26(8):1352–1394, December 2016. doi:10.1017/S0960129514000619.

13 Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session types for access and information flow control. In *21th International Conference on Concurrency Theory (CONCUR)*, pages 237–252, 2010. doi:10.1007/978-3-642-15375-4_17.

14 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Aspects of Computing*, 28(4):669–696, 2016. doi:10.1007/s00165-016-0381-3.

15 Silvia Crafa, Michele Bugliesi, and Giuseppe Castagna. Information flow security for boxed ambients. *Electronic Notes in Theoretical Computer Science*, 66(3):76–97, 2002. doi:10.1016/S1571-0661(04)80417-1.

16 Silvia Crafa and Sabina Rossi. A theory of noninterference for the π-calculus. In *International Symposium on Trustworthy Global Computing (TGC)*, volume 3705 of *Lecture Notes in Computer Science*, pages 2–18. Springer, 2005. doi:10.1007/11580850_2.

17 Silvia Crafa and Sabina Rossi. P-congruences as non-interference for the pi-calculus. In *ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 13–22. ACM, 2006. doi:10.1145/1180337.1180339.

18 Silvia Crafa and Sabina Rossi. Controlling information release in the π-calculus. *Information and Computation*, 205(8):1235–1273, August 2007. doi:10.1016/j.ic.2007.01.001.

19 Ornela Dardha and Simon J. Gay. A New Linear Logic for Deadlock-Free Session-Typed Processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures*, Lecture Notes in Computer Science, pages 91–109. Springer International Publishing, 2018. doi:10.1007/978-3-319-89366-2_5.

20 Farzaneh Derakhshan, Stephanie Balzer, and Limin Jia. Session logical relations for noninterference. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–14. IEEE Computer Society, 2021. doi:10.1109/LICS52264.2021.9470654.

21 Henry DeYoung, Frank Pfenning, and Klaas Pruiksma. Semi-axiomatic sequent calculus. In *5th International Conference on Formal Structures for Computation and Deduction (FSCD)*, volume 167 of *LIPIcs*, pages 29:1–29:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSCD.2020.29.

22 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. In *15th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 143–156. ACM, 2010. doi:10.1145/1863543.1863566.

23 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. *Journal of Functional Programming*, 22(4-5):477–528, 2012. doi:10.1017/S095679681200024X.

24 Daniel Hedin and Andrei Sabelfeld. A Perspective on Information-Flow Control. In *Software Safety and Security*, pages 319–347. IOS Press, 2012. doi:10.3233/978-1-61499-028-4-319.

25 Matthew Hennessy. The security pi-calculus and non-interference. *The Journal of Logic and Algebraic Programming*, 63(1):3–34, April 2005. doi:10.1016/j.jlap.2004.01.003.

26 Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems*, 24(5):566–591, September 2002. doi:10.1145/570886.570890.

27 Bas van den Heuvel, Farzaneh Derakhshan, and Stephanie Balzer. Information Flow Control in Cyclic Process Networks, July 2024. doi:10.48550/arXiv.2407.02304.

28 Bas van den Heuvel, Joseph W. N. Paulus, Daniele Nantes-Sobrinho, and Jorge A. Pérez. Typed Non-determinism in Functional and Concurrent Calculi. In Chung-Kil Hur, editor, *Programming Languages and Systems*, Lecture Notes in Computer Science, pages 112–132, Singapore, 2023. Springer Nature. doi:10.1007/978-981-99-8311-7_6.

29 Bas van den Heuvel and Jorge A. Pérez. Deadlock freedom for asynchronous and cyclic process networks. In Julien Lange, Anastasia Mavridou, Larisa Safina, and Alceste Scalas, editors, *Proceedings 14th Interaction and Concurrency Experience, Online, 18th June 2021*, volume 347 of *Electronic Proceedings in Theoretical Computer Science*, pages 38–56. Open Publishing Association, 2021. doi:10.4204/EPTCS.347.3.

30 Bas van den Heuvel and Jorge A. Pérez. A decentralized analysis of multiparty protocols. *Science of Computer Programming*, page 102840, June 2022. doi:10.1016/j.scico.2022.102840.

31 C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

32 Kohei Honda. Types for dyadic interaction. In *4th International Conference on Concurrency Theory (CONCUR)*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

33 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *7th European Symposium on Programming (ESOP)*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

34 Kohei Honda, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In *9th European Symposium on Programming (ESOP)*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199. Springer, 2000. doi:10.1007/3-540-46425-5_12.

35 Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. In *29th SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 81–92. ACM, 2002. doi:10.1145/503272.503281.

36 Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(6):31, 2007. doi:10.1145/1286821.1286822.

37 Chung-Kil Hur and Derek Dreyer. A kripke logical relation between ML and assembly. In *38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 133–146. ACM, 2011. doi:10.1145/1926385.1926402.

38 Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. In *8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 128–141, 2001. doi:10.1145/360204.360215.

39 Atsushi Igarashi and Naoki Kobayashi. A generic type system for the pi-calculus. *Theoretical Computer Science*, 311(1-3):121–163, 2004. doi:10.1016/S0304-3975(03)00325-6.

40 Naoki Kobayashi. A partially deadlock-free typed process calculus. In *12th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 128–139. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614941.

41 Naoki Kobayashi. Type-based information flow analysis for the pi-calculus. *Acta Informatica*, 42(4-5):291–347, 2005. doi:10.1007/s00236-005-0179-x.

42 Naoki Kobayashi. A New Type System for Deadlock-Free Processes. In Christel Baier and Holger Hermanns, editors, *CONCUR 2006 – Concurrency Theory*, Lecture Notes in Computer Science, pages 233–247. Springer Berlin Heidelberg, 2006. doi:10.1007/11817949_16.

43 Wen Kokke, J. Garrett Morris, and Philip Wadler. Towards races in linear logic. *Logical Methods in Computer Science*, 16(4), 2020. URL: https://lmcs.episciences.org/6979.

44 Vasileios Koutavas and Mitchell Wand. Small bisimulations for reasoning about higher-order imperative programs. In *33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 141–152. ACM, 2006. doi:10.1145/1111037.1111050.

45 Sam Lindley and J. Garrett Morris. A semantics for propositions as sessions. In *24th European Symposium on Programming (ESOP)*, volume 9032 of *Lecture Notes in Computer Science*, pages 560–584. Springer, 2015. doi:10.1007/978-3-662-46669-8_23.

46 Sam Lindley and J. Garrett Morris. Talking bananas: Structural recursion for session types. In *21st ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 434–447. ACM, 2016. doi:10.1145/2951913.2951921.

47 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. doi:10.1007/3-540-10235-3.

48 Robin Milner. *Communication and Concurrency*. PHI Series in Computer Science. Prentice Hall, 1989.

49 Robin Milner. *Communicating and Mobile Systems - the Pi-calculus*. Cambridge University Press, 1999.

50 Georg Neis, Derek Dreyer, and Andreas Rossberg. Non-parametric parametricity. *Journal of Functional Programming*, 21(4-5):497–562, 2011. doi:10.1017/S0956796811000165.

51 Luca Padovani. Deadlock and Lock Freedom in the Linear π-calculus. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, CSL-LICS '14, pages 72:1–72:10, New York, NY, USA, 2014. ACM. doi:10.1145/2603088.2603116.

52 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations for session-based concurrency. In *21st European Symposium on Programming (ESOP)*, volume 7211 of *Lecture Notes in Computer Science*, pages 539–558. Springer, 2012. doi:10.1007/978-3-642-28869-2_27.

53 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Information and Computation*, 239:254–302, 2014. doi:10.1016/j.ic.2014.08.001.

54 Andrew M. Pitts and Ian Stark. Operational reasoning for functions with local state. *Higher Order Operational Techniques in Semantics (HOOTS)*, pages 227–273, 1998.

55 F. Pottier. A simple view of type-secure information flow in the π-calculus. In *Proceedings 15th IEEE Computer Security Foundations Workshop (CSFW-15)*, pages 320–330, 2002. doi:10.1109/CSFW.2002.1021826.

56 Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. *Proceedings of the ACM on Programming Languages*, 5(ICFP):1–31, 2021. doi:10.1145/3473567.

57 Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal of Selected Areas in Communications*, 21(1):5–19, 2003. doi:10.1109/JSAC.2002.806121.

58 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *22nd Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 293–302. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.17.

59 Davide Sangiorgi and David Walker. *The Pi-Calculus - a Theory of Mobile Processes*. Cambridge University Press, 2001.

60 Geoffrey Smith and Dennis M. Volpano. Secure information flow in a multi-threaded imperative language. In *25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 355–364. ACM, 1998. doi:10.1145/268946.268975.

61 Kristian Støvring and Søren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In *34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 161–172. ACM, 2007. doi:10.1145/1190216.1190244.

62 Eijiro Sumii and Benjamin C. Pierce. A bisimulation for type abstraction and recursion. *Journal of the ACM*, 54(5):26, 2007. doi:10.1145/1284320.1284325.

63 Jacob Thamsborg and Lars Birkedal. A kripke logical relation for effect-based program transformations. In *16th ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 445–456. ACM, 2011. doi:10.1145/2034773.2034831.

64 Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *22nd European Symposium on Programming (ESOP)*, volume 7792 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2013. doi:10.1007/978-3-642-37036-6_20.

65 Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996. doi:10.3233/JCS-1996-42-304.

66 Philip Wadler. Propositions as sessions. In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, pages 273–286. ACM, 2012. doi:10.1145/2364527.2364568.

67 Philip Wadler. Propositions as sessions. *Journal of Functional Programming*, 24(2-3):384–418, May 2014. doi:10.1017/S095679681400001X.

68 Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003), 30 June - 2 July 2003, Pacific Grove, CA, USA*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.