Securing Interactive Programs

Willard Rafnsson Daniel Hedin Andrei Sabelfeld
Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden

***Abstract*—This paper studies the foundations of information-flow security for interactive programs. Previous research assumes that the environment is total, that is, it must always be ready to feed new inputs into programs. However, programs secure under this assumption can leak the presence of input. Such leaks can be magnified to whole-secret leaks in the concurrent setting. We propose a framework that generalizes previous research along two dimensions: first, the framework breaks away from the totality of the environment and, second, the framework features fine-grained security types for communication channels, where we distinguish between the security level of message presence and message content. We show that the generalized framework features appealing compositionality properties: parallel composition of secure program results in a secure thread pool. We also show that modeling environments as strategies leads to strong compositionality: various types of composition (with and without scoping) follow from our general compositionality result. Further, we propose a type system that supports enforcement of security via fine-grained security types.**

I. Introduction

Motivation: Is program $\texttt{in}_H(x); \texttt{out}_L(1)$ secure? This program receives an input on a secret (*high*-confidentiality) channel H, stores it in variable x, and outputs constant 1 on a public (*low*-confidentiality) channel L. Upon observing the low output, the attacker can deduce that a high input has been received. Hence, the *presence* of high input is revealed (but not its value). This kind of leak is often undesirable. For example, whether or not any communication with a medical web site takes place in a given browser tab should not be revealed to any web sites that are opened in the other tabs of the browser.

Further, this leak can be magnified in the presence of concurrency. Consider the following two programs:

$$\texttt{in}_{H_0}(x); \texttt{out}_L(0) \quad \text{and} \quad \texttt{in}_{H_1}(x); \texttt{out}_L(1)$$

where both H_0 and H_1 are high channels. Say these programs are run in parallel with the following thread:

$$\texttt{in}_H(x); \texttt{if } x \texttt{ then out}_{H_1}(1) \texttt{ else out}_{H_0}(1)$$

First, a secret value is received and stored in the variable x. Depending on its value, a message is sent on either H_1 or H_0. The parallel composition (where the scope of H_0 and H_1 is made internal to the thread pool) is obviously insecure. Indeed, the parallel composition leaks whether or not the secrets entered on the high channel H is zero. The result of the leak is sent out on the low channel L. It is straightforward to turn this example into a whole-secret leak by wrapping the above threads into loops. The last thread can then walk through the bits of a secret and the first two threads output the bits on the public channel. (A similar example can be constructed with a single high channel.)

$$\begin{array}{l} \texttt{while } 1 \texttt{ do } (\texttt{in}_{H_0}(x); \texttt{out}_L(0)) \\ \mid\mid \texttt{while } 1 \texttt{ do } (\texttt{in}_{H_1}(x); \texttt{out}_L(1)) \\ \mid\mid \texttt{in}_H(h); \\ \quad \texttt{for } b \texttt{ in } bits(h) \texttt{ do} \\ \quad\quad \texttt{if } b \texttt{ then out}_{H_1}(1) \texttt{ else out}_{H_0}(1) \end{array} \tag{1}$$

As simple as this, the example points out a gap in the research on security for interactive programs. Clearly, when the presence of messages is secret, the program $\texttt{in}_H(x); \texttt{out}_L(1)$ is leaky. Surprisingly, the state-of-the-art in security for interactive programs [OCC06], [CH08] imposes the assumption of *totality* of the environment, prescribing that the environment must always be ready to feed new inputs into programs. (Note that totality of environments is not to be confused with the notion of *input totality* [McC87], which requires that a system in any state can accept an input.)

On the other side of the spectrum is work that distinguishes between the security of message presence and content [SM02], [AHS08], [RS11]. Such a distinction is important for modeling encryption, where the observation of a ciphertext reveals message presence but not its original content. However, an unaddressed limitation of this work is that it does not model full interaction with users or programs, but simply assumes all input is precomputed and provided by *streams* of values. In addition, there is also work that neither models strategies nor protects the presence of secret input [AHS08], [AS09], [BPS+09].

We propose a framework that bridges this gap and generalizes previous research along two dimensions: breaking away from the totality of the environment and featuring fine-grained security types for communication channels. The rest of this section provides background on the security of interactive programs, positions our contributions, and overviews the results contained in the paper.

Background: We study reactive programs that are capable of consuming input from the environment, perform internal computation, and produce output to the environment.

Despite a body of work on interaction in process calculi [FG95], [RS99], [HVY00], [Rya01], [HY02], [Pot02], [Kob03] and event-based systems [Man00], [Man01], [SM02], relatively little has been done on tracking the flow of information through language constructs in interactive languages. The state of the art in the area of security for interactive programs is primarily the work by O'Neill et al. [OCC06] and Clark and Hunt [CH08]. O'Neill et al. argue for the need of direct reasoning on the security of interactive programs, to complement the body of work on the security of interactive systems in general. Inspired by Wittbold and Johnson's nondeducibility on strategies [WJ90], O'Neill et al. investigate the security of interactive programs in the presence of user strategies. A key example that shows intricacies of reasoning about interactive programs originates from Wittbold and Johnson:

$$\begin{array}{l}\texttt{while } 1 \texttt{ do } (\\ \quad x := 0 \,[]\, x := 1;\\ \quad \texttt{out}_H(x);\\ \quad \texttt{in}_H(y);\\ \quad \texttt{out}_L(x \oplus y))\end{array} \tag{2}$$

where [] is nondeterministic choice. Assume the program operates on binary values. Given an observation of low output, any high input is consistent on the high channel. However, the environment can make the program propagate a secret value z to the low channel. All the environment needs to do is to take the output (value of x) and xor it with z and provide $x \oplus z$ as input. This example motivates the need to reason about security of programs in the presence of *strategies* [WJ90]. Clark and Hunt [CH08] focus on reducing the security for interactive programs to the security of programs that operate on streams of inputs (without feedback). They prove that it makes no difference in a deterministic setting whether the environment is represented by strategies or streams.

Let us draw closer attention to the interactive setting of the previous work [OCC06], [CH08]. Programs interact with strategies using communication channels. The strategies are functions that, given a trace that is observed on a certain channel (or generally, a set of channels at a given observation level), produce a value that serves as the next input for the program. A critical assumption in this work is the totality of the environment, which demands that strategies must always be able to produce new inputs: there is no way for the environment to block the program by not supplying an input, as demonstrated earlier.

We argue that the assumption of totality limits the space of possible attacks in an undesirable way. Recall the program $\texttt{in}_H(x); \texttt{out}_L(1)$. This program is considered secure in [OCC06], [CH08]. However, when the environment has the possibility of providing or not providing an input on the secret channel, the program is clearly insecure. The output on the low channel leaks the (one-bit) information about whether or not an input is provided on the high channel.

Further, the impact of totality impedes on the compositionality of security definitions. Recall the example with the three threads that opens this section. Similarly to the initial program, these programs are considered secure. Yet their parallel composition leaks high information on the low channel.

Consider a possible modification of the previous models to mimic nontotal environments by providing a special "no further input available" value. With such a modification, program $\texttt{in}_H(x); \texttt{out}_L(1)$ can be ruled out as insecure, provided that the input construct adequately treats the "no further input available" value by blocking or crashing. We choose to explicitly model the possibility for the environment to block the program, so that we do not need to modify the semantics for input.

Contributions: This paper presents a generalized framework for security via strategies, where the totality assumption is dropped: the framework includes both total and nontotal strategies. Further, we parametrize our policies in the level of message presence. In other words, we distinguish between the security level of message presence (existence) and message content.

We illustrate that our generalized framework does not break the relation to deterministic strategies and streams from the work by Clark and Hunt for total strategies. Indeed, we are able to "replay" the results by Clark and Hunt (summarized below).

We show that the generalized framework features appealing compositionality properties: parallel composition of secure program results in a secure thread pool. The power of strategies gives us strong compositionality: we illustrate that various types of composition (with and without scoping) follow from our general compositionality result.

Finally, we provide a type system to illustrate how security can be enforced for our framework. The fine-grained types distinguish between the levels of message presence and content and provide elegant rules for typing the parallel composition.

Overview: Our results, combined with the results from previous work, form the following big picture, displayed by the diagram in Figure 1. The leftmost column in the diagram comes from Clark and Hunt's work [CH08]. The diagram relates sets of programs that correspond to the security conditions and the type system. As we walk through the diagram, we will informally introduce the notation for the sets. This notation is formalized later in the paper.

Our main security condition is strategy-based noninterference. The set of secure programs according to this condition is $\mathbf{Strat}$-NI. The first series of results (presented in

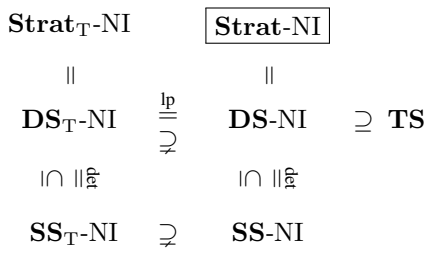

Figure 1. Overview

Section III) positions our condition with respect to other strategy-based definitions. We show that it makes no difference for the security of a program whether the definition only considers deterministic strategies (which corresponds to the set **DS**-NI). This generalizes the result for total strategies [CH08]. We establish that our condition captures attacks that are not captured by programs $\mathbf{Strat}_{\mathrm{T}}$-NI secure under total strategies. Hence the inclusions $\mathbf{DS}_{\mathrm{T}}$-NI $\supsetneq$ **DS**-NI and $\mathbf{Strat}_{\mathrm{T}}$-NI $\supsetneq$ **Strat**-NI. At the same time, we show that our condition is a conservative extension of noninterference on total strategies, in the sense that the definitions coincide if secrets are restricted to travel on channels with low-presence levels (lp).

The series of the results (from Section IV) that corresponds to the last row in the diagram focuses on the relation to stream models. We show that the results in the total setting [CH08] are preserved in our generalization. In particular, it makes no difference for the security of deterministic programs (det) whether we use a strategy or stream-based noninterference.

The next series of results (reported in Section V) is about compositionality (marked by a box in the picture). We show that the totality of the environment impedes on compositionality properties in the previous work. In our setting, we restore compositionality for a general type of thread composition.

The final contribution is a type system (Section VI). Typable programs are secure, hence the **TS** $\subseteq$ **DS**-NI inclusion. The novel rules are for the parallel composition and for the treatment of presence levels on channels. The compositionality results leverage high modularity of the type system.

We proceed by setting up the framework (Section II), followed by the main technical results (Sections III–VI). We discuss related work (Section VII) and wrap the paper up with concluding remarks (Section VIII).

II. Framework

As outlined above, our aim is to secure information flows in interactive programs. We address this issue in an adaption of a standard framework for interactive programs [OCC06], [CH08], [BPS+09], [RS11]. Here, information can only enter and exit programs through channel-based message passing. Each channel comes with a label expressing the *confidentiality* level of the information it carries. We then ensure that confidential information in inputs does not influence which public inputs and outputs the program can perform.

A. Interactive Programs

Inputs i, outputs o, and messages a, last of which we also refer to as (inter)actions, are given by

$$i ::= \alpha?v \qquad o ::= \alpha!v \mid \tau \qquad a ::= i \mid o$$

where $\alpha?v$ (resp. $\alpha!v$) denotes a message received (resp. sent) on channel α carrying value v, and τ denotes a computation step other than an interaction with an environment. Here, α and v respectively range over the (unspecified nonempty) sets $\mathbb{C}$ and $\mathbb{V}$. Channels are the only external interface to our systems, and are therefore the only medium by which information can enter and exit our systems.

Our model of computation is a *labeled transition system (LTS)*. An LTS is a triple $(S, A, \{\xrightarrow{\mathbf{a}} \mid \mathbf{a} \in A\})$, where S is a set of *states*, A a set of *actions*, and for all $\mathbf{a} \in A$, $\xrightarrow{\mathbf{a}} \subseteq S \times S$. We write $s \xrightarrow{\mathbf{a}}$ when $s \xrightarrow{\mathbf{a}} s'$ for some s'. The behavior of an interactive program can be given as an LTS as follows.

Definition II.1. An *input-output LTS* (IOLTS) is a LTS, with A ranged by a, which is *input-neutral*, that is for all $s \in S$, if $\exists v.\, s \xrightarrow{\alpha?v}$, then $\forall v.\, s \xrightarrow{\alpha?v}$.

Intuitively, if s is an IOLTS state which can, as its next computation step, input i and enter state s', then $s \xrightarrow{i} s'$. Likewise, $s \xrightarrow{o} s'$ if s can output o and enter s'. Practical computation models native to this paradigm include event loops and programs written in Erlang and JavaScript. Here, states correspond to program configurations, that is, code (and possibly a code pointer) paired with its environment, and actions express an interaction of a running program with its context. Section VI demonstrates how to give the semantics of programs written in a simple nondeterministic imperative programming language as an IOLTS.

Definition II.2. An IOLTS is deterministic iff

1) If $s \xrightarrow{a_1} s_1$, $s \xrightarrow{a_2} s_2$ and $a_1 \neq a_2$, then $a_1 = \alpha?v_1$ and $a_2 = \alpha?v_2$ for some α, v_1 and v_2.
2) If $s \xrightarrow{a} s_1$, $s \xrightarrow{a} s_2$, then $s_1 = s_2$.

Pt. 1) says that if $s \xrightarrow{\alpha?v}$, then $s \xrightarrow{a}$ iff $a \in \{\alpha?v \mid v \in \mathbb{V}\}$, and implicitly, if $s \xrightarrow{o}$, then $s \xrightarrow{a}$ iff $a = o$. Pt. 2) says that s has no internal nondeterminism.

Let Tr denote the set a^* of *traces*, ranged by t. We write $s \xrightarrow{t} s'$ when we have $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s_n$ for some $t = a_1.\cdots.a_n$ and $s_1, \ldots, s_n$ with $s_n = s'$. We let $t \restriction_?$, $t \restriction_!$ and $t \restriction_\alpha$ denote the inputs, outputs and α-messages, in t, respectively. That is, if $t = \alpha?0.\alpha'!1.\alpha'?2.\alpha!3$, then $t\restriction_? = \alpha?0.\alpha'?2$, $t\restriction_! = \alpha'!1.\alpha!3$ and $t\restriction_\alpha = \alpha?0.\alpha!3$. We write $t \leq t''$ when, for some t', $t'' = t.t'$.

B. Observables

The observables of an interactive program are its inputs and outputs. Whether a message on a channel is observable or not is indicated by the *security levels* associated with the channel. We assume a lattice of security levels $(\mathcal{L}, \sqsubseteq)$, with $\mathcal{L}$ ranged by l, expressing levels of confidentiality. In our examples, $\mathcal{L} = \{L, H\}$ and $\sqsubseteq = \{(L,L),(L,H),(H,H)\}$, H for "high" and L for "low" confidentiality. The channel-to-levels labeling is denoted $\gamma : \mathbb{C} \to \sqsubseteq$. Here, if $\gamma(\alpha) = l_1^{l_2}$, with $l_1^{l_2}$ abbreviating (l_2, l_1), then l_1 is the confidentiality level of values (content) passed on α, and l_2 the confidentiality level of the presence of a message on α. In examples, we will frequently represent a channel by its security label. We abbreviate L^L, H^L and H^H by L, M and H, respectively. No observer can see τ actions.

The security labels express *who* can observe *what*. An observer is associated a security level l, indicating that the observer is capable of observing values on α if $l_1 \sqsubseteq l$, and the presence of messages on α if $l_2 \sqsubseteq l$, where $\gamma(\alpha) = l_1^{l_2}$. We denote the l-observables in t by $t \restriction_l$. For $t = \mathsf{L?0.H!1.M?2.L!3}$, $t \restriction_L = \mathsf{L?0.M?}\,\square\,\mathsf{.L!3}$. Here, $\square \notin \mathbb{V}$ is a "blank", representing an unobservable value.

Definition II.3. t_1 and t_2 are *l-equivalent*, written $t_1 =_l t_2$, iff $t_1 \restriction_l = t_2 \restriction_l$.

So, l-equivalent traces are observably equivalent to an l-observer. We write $t \leq_l t''$ if $t =_l t'$ for some $t' \leq t''$.

C. Strategies

The inputs to our systems come from the environment in which the system runs. The environment might vary the input on a channel α depending on which interaction trace t the environment has observed. Further, the environment might pick an input value nondeterministically, or not input any value at all. We model an environment as a mapping from interaction traces t and channels α to the (possibly empty) set of values V from which the environment draws a value to input on request on α after observing t. To ensure that an observer can attribute observably different interaction traces to a leak in the interactive program, we place two restrictions on the environments we take into consideration in our framework. First, if the observer can observe values passed on α, then the environment must be defined the same way on α for all observably equivalent traces. Second, if the observer can observe the presence of messages passed on α, then the environment must, for each set of observably equivalent traces, map all of them to (possibly different) values, or map none of them to a value[1]. We refer to these environments as *strategies*.

[1]Not considering strategies which, given an interaction trace t and an input request on channel α, either return a value, or no value, does not affect our security results (as our policy is possibilistic).

Definition II.4. A *strategy* is a function $\omega : \mathbb{C} \to \mathrm{Tr} \to \mathcal{P}(\mathbb{V})$ such that, for all α; $\gamma(\alpha) = l_1^{l_2}$, with ω_α denoting $\omega(\alpha)$,

$$t_1 =_{l_1} t_2 \implies \omega_\alpha(t_1) = \omega_\alpha(t_2)$$
$$t_1 =_{l_2} t_2 \implies \omega_\alpha(t_1) \doteq \omega_\alpha(t_2).$$

Here, $A \doteq B$ is defined as $A = \emptyset \iff B = \emptyset$, and satisfies the following property.

$$A = B \implies A \doteq B. \tag{3}$$

Let $\mathbf{Strat}$ denote the set of strategies. A strategy which always inputs on α the number of past α-outputs can be defined as $\omega_\alpha(t) = |t \restriction_\alpha \restriction_!|$.

Definition II.5. ω and ω' are *l-equivalent*, written $\omega =_l \omega'$, iff, for all α; $\gamma(\alpha) = l_1^{l_2}$ and t,

$$l_1 \sqsubseteq l \implies \omega_\alpha(t) = \omega'_\alpha(t)$$
$$l_2 \sqsubseteq l \implies \omega_\alpha(t) \doteq \omega'_\alpha(t).$$

It is instructive to look at strategies as restrictions on which traces are possible; t is *consistent* with ω, written $\omega \models t$, iff for all $\alpha?v$, t' and t'' for which $t = t'.\alpha?v.t''$, $v \in \omega_\alpha(t')$. Running a system under a strategy thus constrains the traces which the system can perform, as system inputs must come from the strategy; *s produces t under ω*, written $\omega \models s \xrightarrow{t}$, iff $s \xrightarrow{t}$ and $\omega \models t$.

D. Noninterference

Our security policy of interest is that of *possibilistic noninterference* from [CH08], which is a generalization of Definition 1 from [OCC06]. The policy states that under observably equivalent strategies, drawn from $W \subseteq \mathbf{Strat}$, the respective sets of traces s produces under either of them are observably equivalent.

Definition II.6. s is *W-noninterfering* iff

$$\begin{aligned} &\forall l\,.\,\forall \omega_1, \omega_2 \in W\,.\,\omega_1 =_l \omega_2 \implies \\ &\forall t_1\,.\,\omega_1 \models s \xrightarrow{t_1} \implies \\ &\exists t_2\,.\,\omega_2 \models s \xrightarrow{t_2} \wedge\ t_1 =_l t_2. \end{aligned} \tag{NI}$$

The larger W is, the larger the space of attacks a W-noninterfering s is protected from.

Definition II.7. A *W-attack* is a 4-tuple $(l, \omega_1, \omega_2, t_1)$ where $\omega_1, \omega_2 \in W$, $\omega_1 =_l \omega_2$ and $\omega_1 \models t_1$. It is an *attack on s* iff

1) $\omega_1 \models s \xrightarrow{t_1}$, and
2) $\forall t_2\,.\,\omega_2 \models s \xrightarrow{t_2} \implies t_2 \neq_l t_1$.

It is easy to see that s is W-noninterfering iff there is no W-attack on s. Let W-NI denote the set of W-noninterfering programs. We say s is noninterfering iff $s \in \mathbf{Strat}$-NI.

We get the following lemma from Definition II.6, since a W_1-attack on s is a W_2-attack on s, for any $W_1 \subseteq W_2$.

Lemma II.8. *For all $W_1, W_2 \subseteq \mathbf{Strat}$, we have*

$$W_1 \subseteq W_2 \implies W_2\text{-NI} \subseteq W_1\text{-NI}.$$

III. Generalized strategies for noninterference

We now study the interplay between interactive programs, channel labelings and strategies.

A. Total Strategies

First, we contrast **Strat** with the total strategies considered in [CH08], a subset of which is considered in [OCC06].

Definition III.1. ω is *total* iff $\forall \alpha, t \,.\, \omega_\alpha(t) \neq \emptyset$.

Let W_T denote the set of total strategies in W. The set of all total strategies is thus $\mathbf{Strat}_\mathrm{T}$. As outlined in the introduction, programs which are protected against $\mathbf{Strat}_\mathrm{T}$-attacks may still have **Strat**-attacks.

Theorem III.2. $\mathbf{Strat}\text{-NI} \subsetneq \mathbf{Strat}_\mathrm{T}\text{-NI}$.

Proof: Lemma II.8 gives $\mathbf{Strat}\text{-NI} \subseteq \mathbf{Strat}_\mathrm{T}\text{-NI}$. *We now show the existence of a program in* $\mathbf{Strat}_\mathrm{T}\text{-NI}$ *which is not in* **Strat**-NI. *Program* $\texttt{in}_\mathsf{H}\ x\texttt{;}\ \texttt{out}_\mathsf{L}\ 0$ *is such a program. It is in* $\mathbf{Strat}_\mathrm{T}\text{-NI}$ *since totality of strategies in* $\mathbf{Strat}_\mathrm{T}$ *gives*

$$\forall \omega \in \mathbf{Strat}_\mathrm{T} \,.\, \exists v \,.\, \omega \models s \xrightarrow{\mathsf{H}?v},$$

and thus

$$\forall \omega \in \mathbf{Strat}_\mathrm{T} \,.\, \exists v \,.\, \omega \models s \xrightarrow{\mathsf{H}?v.\mathsf{L}!0}.$$

So (NI) *holds with l instantiated to L.* (NI) *with l instantiated to H follows from $\omega_1 =_H \omega_2 \implies \omega_1 = \omega_2$. However, this program is not* **Strat**-NI. *In particular, consider*

$$\omega_{1_\alpha}(t) = \begin{cases} \{42\} & \text{, if } \alpha = \mathsf{H} \\ \emptyset & \text{, otherwise} \end{cases} \qquad \omega_{2_\alpha}(t) = \emptyset$$

Clearly, $\omega_1 =_L \omega_2$. However,

$$\omega_1 \models s \xrightarrow{\mathsf{H}?42.\mathsf{L}!0}$$

and

$$\forall t \,.\, \omega_2 \models s \xrightarrow{t} \implies t = \epsilon \neq_L \mathsf{H}?42.\mathsf{L}!0.$$

Thus $(L, \omega_1, \omega_2, \mathsf{H}?42.\mathsf{L}!0)$ is a **Strat**-*attack on this program. This program is thus not in* **Strat**-NI. ∎

It is worth noting at this point that if we consider the class of channel labelings where the presence and content of messages are labeled with the same security level,

$$\mathrm{img}(\gamma) = \{l^l \mid l \in \mathcal{L}\},$$

then by (3), the definition of strategies and l-equivalence becomes the same as the one given in [CH08] and [OCC06]. Furthermore, $\mathbf{Strat}_\mathrm{T}$-noninterference becomes the same policy as the one given in Definition 8 in [CH08] and, for a subset $\mathbf{Strat}_\mathrm{N}$ of $\mathbf{Strat}_\mathrm{T}$ (called "narrow" strategies in [CH08]) and for deterministic programs, $\mathbf{Strat}_\mathrm{N}$-noninterference becomes the same policy as the one given in Definition 1 in [OCC06]. Since $\texttt{in}_\mathsf{H}\ x\texttt{;}\ \texttt{out}_\mathsf{L}\ 0$ has no $\mathbf{Strat}_\mathrm{T}$-attacks and is deterministic, it is secure according to both these policies.

B. Deterministic Strategies

We have just witnessed the existence of a program which has no $\mathbf{Strat}_\mathrm{T}$-attacks, but which has **Strat**-attacks, meaning that there are interesting attacks in the space between $\mathbf{Strat}_\mathrm{T}$- and **Strat**-attacks. We now take a closer look at this space, characterizing a small subset of it as being of interest. We first consider deterministic strategies.

Definition III.3. ω is *deterministic* iff $\forall \alpha, t \,.\, |\omega_\alpha(t)| \leq 1$.

Let **DS** denote the set of deterministic strategies. We sometimes write $\omega_\alpha(t) = \bot$ instead of $\omega_\alpha(t) = \emptyset$ when ω is deterministic. [CH08] shows that when considering whether a s is protected against $\mathbf{Strat}_\mathrm{T}$-attacks or not, it is sufficient to consider $\mathbf{DS}_\mathrm{T}$-attacks; this is Theorem 1 therein.

Proposition III.4 ([CH08])**.** $\mathbf{Strat}_\mathrm{T}\text{-NI} = \mathbf{DS}_\mathrm{T}\text{-NI}$.

It turns out that the same holds for strategies in general.

Theorem III.5. $\mathbf{Strat}\text{-NI} = \mathbf{DS}\text{-NI}$.

Proof: Same as proof of Theorem 1 in [CH08], as totality is never invoked in the proof. ∎

This theorem rules out the need to take into consideration attacks which utilize nondeterminism to cause a leak. However, the theorem does not make clear which abilities the attacker *must* have in order to create a leak in interactive programs which are not in **Strat**-NI. For instance, do some programs *need* an infinite supply of input, perhaps only on some channels and not others, to leak? Do we need to take into consideration strategies which "discriminate" against some interaction traces by, say, providing input only if an even number of L-outputs has occurred prior?

The answer to both of these questions is no. We only need to consider those **DS**-attacks $(l, \omega_1, \omega_2, t_1)$ where ω_{1_α} and ω_{2_α} always feed α-input on request, as long as the interaction trace has fewer α-inputs than t_1 has. Furthermore, ω_2 does not need to feed input on channels with $\not\sqsubseteq l$ presence at all. The proof of this can be found in the appendix.

Lemma III.6. *If $(l, \omega_1, \omega_2, t_1)$ is a* **Strat**-*attack on s, then for some ω_1' and ω_2', where for all t and $\alpha; \gamma(\alpha) = l_1^{l_2}$,*

$$(\,|t\restriction_\alpha\restriction_?| < |t_1\restriction_\alpha\restriction_?| \iff \omega'_{1_\alpha}(t) \neq \emptyset\,),$$
$$l_2 \sqsubseteq l \implies (\,|t\restriction_\alpha\restriction_?| < |t_1\restriction_\alpha\restriction_?| \iff \omega'_{2_\alpha}(t) \neq \emptyset\,), \text{ and}$$
$$l_2 \not\sqsubseteq l \implies \omega'_{2_\alpha}(t) = \emptyset,$$

$(l, \omega_1', \omega_2', t_1)$ is a **DS**-*attack on s.*

C. Public Presence Labels

The presence of an input on a channel with a secret presence label has a significant impact on the space of attacks to be considered to determine whether an interactive

program is noninterfering or not. Consider for instance the class of channel labelings where message presence is always public. We refer to these γ as lp labelings.

$$\mathrm{img}(\gamma) = \{l^{\perp} \mid l \in \mathcal{L}\}.$$

It turns out that when a program is labeled with such a labeling, then it is sufficient to consider only $\mathbf{Strat}_{\mathrm{T}}$-attacks to determine whether an interactive program is in $\mathbf{Strat}$-NI or not. Before proving this claim, we establish two simple, but useful, lemmas. The first lemma states that when $\omega_1 =_l \omega_2$, then for any α with l-observable presence, ω_{1_α} and ω_{2_α} will be (un)defined on exactly the same traces.

Lemma III.7. *If $\omega_1 =_l \omega_2$, then for all t and α; $\gamma(\alpha) = l_1^{l_2}$, if $l_2 \sqsubseteq l$, then $\omega_{1_\alpha}(t) = \emptyset \iff \omega_{2_\alpha}(t) = \emptyset$.*

Proof: Follows from (3) *and Definition II.5.* ∎

It is never the case that an attack works as a consequence of one strategy supplying input on a channel with observable presence, and the other strategy not doing so (on an observably equivalent trace).

Lemma III.8. *For all s, $\mathbf{Strat}$-attacks $(l, \omega_1, \omega_2, t_1)$ on s, $\alpha; \gamma(\alpha) = l_1^{l_2}$ and t, if $l_2 \sqsubseteq l$, then*

$$\forall v \,.\, \omega_2 \models s \xrightarrow{t} s' \wedge s' \xrightarrow{\alpha?v} \wedge \omega_{2_\alpha}(t) = \emptyset \implies t.\alpha?v \not\leq_l t_1 \tag{4}$$

Proof: Let s, and $\mathbf{Strat}$-attack $(l, \omega_1, \omega_2, t_1)$ on s, be given. By Definition II.7 Pt. 1), we have $\forall t, \alpha, v$,

$$t.\alpha?v \leq t_1 \implies v \in \omega_{1_\alpha}(t) \neq \emptyset.$$

By Definition II.4 we get $\forall t, t', \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l$,

$$t =_{l_2} t' \implies (\omega_{j_\alpha}(t) = \emptyset \iff \omega_{j_\alpha}(t') = \emptyset).$$

Since $\omega_1 =_l \omega_2$, Lemma III.7 yields $\forall t, \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l$,

$$\omega_{1_\alpha}(t) = \emptyset \iff \omega_{2_\alpha}(t) = \emptyset.$$

Together, this gives $\forall t, \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l, v$,

$$t.\alpha?v \leq_l t_1 \implies \omega_{2_\alpha}(t) \neq \emptyset.$$

By contraposition, we get $\forall t, \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l, v$,

$$\omega_{2_\alpha}(t) = \emptyset \implies t.\alpha?v \not\leq_l t_1.$$

(4) *follows by specializing the premise of the implication.* ∎

We are now ready to prove the above-stated claim.

Theorem III.9. $\mathbf{Strat}$-NI $= \mathbf{Strat}_{\mathrm{T}}$-NI *for* lp γ.

Proof: We prove $\mathbf{DS}$-NI $= \mathbf{DS}_{\mathrm{T}}$-NI*; the result will then follow from Theorem III.5 and Proposition III.4.* $\mathbf{DS}_{\mathrm{T}} \subseteq \mathbf{DS}$ *by definition of* $\mathbf{DS}_{\mathrm{T}}$*. By Lemma II.8,* $\mathbf{DS}$-NI $\subseteq \mathbf{DS}_{\mathrm{T}}$-NI*. We now show* $\mathbf{DS}$-NI $\supseteq \mathbf{DS}_{\mathrm{T}}$-NI*, i.e.,*

$$\forall s \,.\, s \in \mathbf{DS}_{\mathrm{T}}\text{-NI} \implies s \in \mathbf{DS}\text{-NI}.$$

We show instead the contrapositive. That is,

$$\forall s \,.\, s \notin \mathbf{DS}\text{-NI} \implies s \notin \mathbf{DS}_{\mathrm{T}}\text{-NI}. \tag{5}$$

Let $s \notin \mathbf{DS}$-NI be given. Then s has some $\mathbf{DS}$-attack $(l, \omega_1, \omega_2, t_1)$, that is, for some l, deterministic ω_1 and ω_2, and t_1,

1) $\omega_1 =_l \omega_2$,
2) $\omega_1 \models s \xrightarrow{t_1}$,
3) $\forall t_2 \,.\, \omega_2 \models s \xrightarrow{t_2} \implies t_2 \neq_l t_1$.

By Lemmas III.7 and III.8, and by lp*, we get $\forall t, \alpha$,*

$$\omega_{1_\alpha}(t) = \perp \iff \omega_{2_\alpha}(t) = \perp \tag{6}$$

and $\forall t, \alpha, v$,

$$\omega_2 \models s \xrightarrow{t} s' \wedge s' \xrightarrow{\alpha?v} \wedge \omega_{2_\alpha}(t) = \perp \implies t.\alpha?v \not\leq_l t_1, \forall v. \tag{7}$$

In particular, this holds if we fix v to a constant k. Let

$$\omega'_{j_\alpha}(t) = \begin{cases} k & \text{, if } \omega_{j_\alpha}(t) = \perp \\ \omega_{j_\alpha}(t) & \text{, otherwise.} \end{cases}$$

We show that $(l, \omega'_1, \omega'_2, t_1)$ is a $\mathbf{DS}_{\mathrm{T}}$-attack on s, that is,

i) $\omega'_1 =_l \omega'_2$
ii) ω'_j *is a deterministic total strategy,*
iii) $\omega'_1 \models s \xrightarrow{t_1}$
iv) $\forall t_2 \,.\, \omega'_2 \models s \xrightarrow{t_2} \implies t_2 \neq_l t_1$.

It is easy to see that $\forall t, \alpha$,

$$\omega_{1_\alpha}(t) = \omega_{2_\alpha}(t) = \perp \implies \omega'_{1_\alpha}(t) = \omega'_{2_\alpha}(t)$$

This, 1) and (6) *gives i). For α for which $\gamma(\alpha) = l_1^{l_2}$, since $t =_{l_2} t' \implies \omega_{j_\alpha}(t) = \perp \iff \omega_{j_\alpha}(t') = \perp$ and $t =_{l_1} t' \implies t =_{l_2} t'$, then either $\omega'_{j_\alpha}(t) = \omega'_{j_\alpha}(t') = k$ or $\omega'_{j_\alpha}(t) = \omega_{j_\alpha}(t)$ and $\omega'_{j_\alpha}(t') = \omega_{j_\alpha}(t')$. Thus, since ω_j are strategies, so are ω'_j. By definition of ω'_j, ω'_j is total. Since ω_j is deterministic, then by definition of ω'_j, so is ω'_j. So ii) holds. It is easy to see that $\forall t, \alpha$,*

$$\omega_{j_\alpha}(t) \neq \perp \implies \omega_{j_\alpha}(t) = \omega'_{j_\alpha}(t)$$

This, and 1), gives iii). By (7)*, iv) holds.*

Thus $s \notin \mathbf{DS}_{\mathrm{T}}$-NI. Since s was arbitrary, (5) *holds.* ∎

As a consequence, programs such as `in`$_{\mathsf{M}}$ x`;` `out`$_{\mathsf{L}}$ `0`, which we already know is $\mathbf{Strat}_{\mathrm{T}}$-noninterfering, are thus $\mathbf{Strat}$-noninterfering, since all interaction occurs on public presence channels, and now flow of message content occurs.

IV. Relation to streams for noninterference

As we saw in (2), in the presence of nondeterminism, an attack $(l, \omega_1, \omega_2, t_1)$ sometimes needs to adapt to observed nondeterministic choices, to then force the program down different control flow paths, to then make a trace t_1 producible under ω_1 not matchable under ω_2. However, as demonstrated in [CH08] Theorem 2, in the $\mathbf{Strat}_{\mathrm{T}}$ setting, if the program in question is deterministic (as many programs

are), there are no nondeterministic choices made by the program for an attack to cleverly adapt to. The space of attacks we need to consider is thus much simpler; it suffices to consider strategies which can be expressed as a (possibly infinite) list of messages on each input channel, independent on interaction traces. These are stream strategies.

Definition IV.1. ω is a *stream strategy* iff it is deterministic and for all α, $|t_1 \restriction_\alpha \restriction_?| = |t_2 \restriction_\alpha \restriction_?| \implies \omega_\alpha(t_1) = \omega_\alpha(t_2)$.

Proposition IV.2 ([CH08])**.** $\mathbf{Strat}_\mathrm{T}$-NI $=$ $\mathbf{SS}_\mathrm{T}$-NI *for deterministic* s.

We prove this result in the **Strat** setting. First we establish an insightful lemma; it says that, when a deterministic program is run under a deterministic strategy, then it produces a unique (possibly infinite) sequence of interactions, of which all interaction traces are prefixes. This is Lemma 4 in [CH08].

Lemma IV.3 ([CH08])**.** *If s and ω are deterministic, then if $\omega \models s \xrightarrow{t_1}$ and $\omega \models s \xrightarrow{t_2}$, then $t_1 \leq t_2$ or $t_2 \leq t_1$.*

Proof: Same as proof of Lemma 4 in [CH08], as totality is never invoked in the proof. ∎

We now prove the above-stated result in the **Strat** setting. The proof borrows the idea from [CH08] of "streamifying" deterministic strategies on the set of traces on which they are defined. Lemma IV.3 then gives us that this change does not affect the set of traces a deterministic program produces under the modified strategies.

Theorem IV.4. $\mathbf{Strat}$-NI $=$ $\mathbf{SS}$-NI *for deterministic* s.

Proof: We prove $\mathbf{DS}$-NI $=$ $\mathbf{SS}$-NI*; the result will then follow from Theorem III.5.* $\mathbf{SS} \subseteq \mathbf{DS}$ *by definition of* $\mathbf{SS}$*. By Lemma II.8,* $\mathbf{DS}$-NI $\subseteq$ $\mathbf{SS}$-NI*. We show* $\mathbf{DS}$-NI $\supseteq$ $\mathbf{SS}$-NI*. That is,*

$$\forall s.\, s \in \mathbf{SS}\text{-NI} \implies s \in \mathbf{DS}\text{-NI}.$$

We show instead the contrapositive. That is,

$$\forall s.\, s \notin \mathbf{DS}\text{-NI} \implies s \notin \mathbf{SS}\text{-NI}.$$

Let $s \notin$ $\mathbf{DS}$-NI be given. Then there is a $\mathbf{DS}$*-attack $(l, \omega_1, \omega_2, t_1)$ on s, that is,*

1) $\omega_1 =_l \omega_2$,
2) $\omega_1 \models s \xrightarrow{t_1}$,
3) $\forall t_2 \,.\, \omega_2 \models s \xrightarrow{t_2} \implies t_2 \neq_l t_1$.

and, by Lemma III.6,

$$\begin{array}{l} |t \restriction_\alpha \restriction_?| < |t_1 \restriction_\alpha \restriction_?| \iff \omega_{1_\alpha}(t) \neq \emptyset, \\ l_2 \sqsubseteq l \implies (\, |t \restriction_\alpha \restriction_?| < |t_1 \restriction_\alpha \restriction_?| \iff \omega_{2_\alpha}(t) \neq \emptyset\,) \\ l_2 \not\sqsubseteq l \implies \omega_{2_\alpha}(t) = \emptyset. \end{array} \quad (8)$$

Let

$$\omega'_{j_\alpha}(t) = \{v \mid \omega_j \models s \xrightarrow{t'.\alpha?v} \text{ for some } t' \text{ with } |t \restriction_\alpha \restriction_?| = |t' \restriction_\alpha \restriction_?|\}$$

We must show that

i) $\omega'_1 =_l \omega'_2$,
ii) ω'_j *is a strategy,*
iii) $\omega'_1 \models s \xrightarrow{t_1}$,
iv) $\omega'_2 \models s \xrightarrow{t_2} \implies t_1 \neq_l t_2$, $\forall t_2$.

For $\alpha = l_1^{l_2}$, since $t =_{l_2} t' \implies |t \restriction_\alpha \restriction_?| = |t' \restriction_\alpha \restriction_?|$ and $t =_{l_1} t' \implies t =_{l_2} t'$, then either $\omega_{j_\alpha}(t) = \omega_{j_\alpha}(t') = \omega'_{j_\alpha}(t) = \omega'_{j_\alpha}(t') = \bot$ or $\omega'_{1_\alpha}(t) = \omega'_{2_\alpha}(t) = V$ for some set $V \neq \emptyset$ of values. Thus, since ω_j are strategies, ii) holds.

Before proceeding with proving i), we show that ω'_j is deterministic, and at the same time a stream strategy. We first show ω'_j is deterministic. Assume the contrary. Then there are some t'_1, t'_2, v_1, v_2 such that

a) $\omega'_j \models s \xrightarrow{t'_1.\alpha?v_1}$, $\omega'_j \models s \xrightarrow{t'_2.\alpha?v_2}$,
b) $|t'_1 \restriction_\alpha \restriction_?| = |t'_2 \restriction_\alpha \restriction_?|$, *and*
c) $v_1 \neq v_2$.

Lemma IV.3 gives $t'_1.\alpha?v_1 \leq t'_2.\alpha?v_2$ or $t'_2.\alpha?v_2 \leq t'_1.\alpha?v_1$. Assume wlg. that $t'_1.\alpha?v_1 \leq t'_2.\alpha?v_2$. Two cases to consider.

$t'_1.\alpha?v_1 < t'_2.\alpha?v_2$: Then $t'_1.\alpha?v_1 \leq t'_2$. But

$$|t'_1.\alpha?v_1 \restriction_\alpha \restriction_?| = |t'_1 \restriction_\alpha \restriction_?| + 1 > |t'_1 \restriction_\alpha \restriction_?| = |t'_2 \restriction_\alpha \restriction_?|\,,$$

contradicting b).

$t'_1.\alpha?v_1 = t'_2.\alpha?v_2$: Then $t'_1 = t'_2$. But now Definition II.2 Pt. 1) gives $v_1 = v_2$, contradicting c).

So ω'_j is deterministic. By definition, ω'_j is also a stream strategy. By (8)*,*

$$\begin{array}{l} |t \restriction_\alpha \restriction_?| < |t_1 \restriction_\alpha \restriction_?| \iff \omega'_{1_\alpha}(t) \neq \emptyset \\ l_2 \sqsubseteq l \implies (\, |t \restriction_\alpha \restriction_?| < |t_1 \restriction_\alpha \restriction_?| \iff \omega'_{2_\alpha}(t) \neq \emptyset\,) \\ l_2 \not\sqsubseteq l \implies \omega'_{2_\alpha}(t) = \emptyset. \end{array} \quad (9)$$

We return to proving i). Let t be arbitrary. To show i), we must show that for all $\alpha; \gamma(\alpha) = l_1^{l_2}$ for which $l_2 \sqsubseteq l$,

a') $\omega_\alpha(t) \doteq \omega'_\alpha(t)$,
b') $l_1 \sqsubseteq l \implies \omega_\alpha(t) = \omega'_\alpha(t)$.

There are three cases to consider.

1') There exists no t' for which we have $|t' \restriction_\alpha \restriction_?| = |t \restriction_\alpha \restriction_?|$, $\omega_1 \models s \xrightarrow{t'.\alpha?v_1}$, $\omega_2 \models s \xrightarrow{t'.\alpha?v_2}$, for any v_1, v_2. Then $\omega'_{1_\alpha}(t) = \omega'_{2_\alpha}(t) = \bot$, satisfying a'), and satisfying b') regardless of whether $l_1 \sqsubseteq l$ holds or not.

2') There exists a t' for which we have $|t' \restriction_\alpha \restriction_?| = |t \restriction_\alpha \restriction_?|$, $\omega_1 \models s \xrightarrow{t'.\alpha?v_1}$, $\omega_2 \models s \not\xrightarrow{t'.\alpha?v_2}$, for some v_1 and all v_2. Then $\omega_{1_\alpha}(t') = v_1$ and $\omega_{2_\alpha}(t') = \bot$, contradicting 1), regardless of whether $l_1 \sqsubseteq l$ holds or not. So a') and b') hold. (the proof of the symmetric case is analogous, so we omit it wlg.)

3') There exists a t' for which we have $|t' \restriction_\alpha \restriction_?| = |t \restriction_\alpha \restriction_?|$, $\omega_1 \models s \xrightarrow{t'.\alpha?v_1}$, $\omega_2 \models s \xrightarrow{t'.\alpha?v_2}$, for some v_1, v_2. Then $\omega'_{1_\alpha}(t) = v_1 \neq \bot$ and $\omega'_{2_\alpha}(t) = v_2 \neq \bot$, satisfying a'). Assume $l_1 \sqsubseteq l$. Then by 1), $\omega_{1_\alpha}(t') = \omega_{2_\alpha}(t') = v$ for some v. Then $v_1 = v$ and $v_2 = v$, so $\omega'_{1_\alpha}(t) = \omega'_{2_\alpha}(t)$.

So i) holds.

It remains to show iii) and iv). To do this, we show

$$\omega_j \models s \xrightarrow{t} \iff \omega'_j \models s \xrightarrow{t} \tag{10}$$

We proceed by induction in $n = |t\restriction_?|$.

$n = 0$**:** *Assume* $\omega_j \models s \xrightarrow{t}$. *Then* $s \xrightarrow{t}$. *Since* $\omega'_j \models t$ *holds vacuously,* $\omega'_j \models s \xrightarrow{t}$ *holds. This proves the forward implication of* (10)*. The proof for the reverse implication is analogous. Thus* (10) *holds.*

$n + 1$**, assuming** n**:** *Assume* (10) *holds* $\forall t$ *with* $|t\restriction_?| = n$. *This is our induction hypothesis (IH). We prove* (10) *for* t *with* $|t\restriction_?| = n+1$. *Let* $t = t'.\alpha?v.t''$ *with* $|t''\restriction_?| = 0$. *Then* $|t'\restriction_?| = n$, *so* (10) *holds with* t *set to* t'.

Assume that $\omega_j \models s \xrightarrow{t}$ *holds. Then* $s \xrightarrow{t}$ *and* $\omega_j \models t$. *Thus for any* $\hat{t}$ *for which* $\hat{t} \leq t$, $s \xrightarrow{\hat{t}}$ *and* $\omega_j \models \hat{t}$. *In particular,* $s \xrightarrow{t'.\alpha?v}$, $\omega_j \models t'.\alpha?v$, *and* $s \xrightarrow{t'}$, $\omega_j \models t'$. *Thus* $\omega_j \models s \xrightarrow{t'.\alpha?v}$ *and* $\omega_j \models s \xrightarrow{t'}$. *From the former, we have* $\omega_{j_\alpha}(t') = v$. *From the latter and by (IH), we get* $\omega'_j \models s \xrightarrow{t'}$. *Since, by definition of* ω'_j, $v \in \omega'_{j_\alpha}(t')$ *and* ω'_j *is deterministic, we get that* $\omega'_{j_\alpha}(t') = v$. *Thus* $\omega'_j \models s \xrightarrow{t'.\alpha?v}$. *Since* $s \xrightarrow{t'.\alpha?v}$, $\omega'_j \models t'.\alpha?v$ *and* $|t''\restriction_?| = 0$, *we get* $s \xrightarrow{t}$, $\omega'_j \models t$, *and thus* $\omega'_j \models s \xrightarrow{t}$. *So the forward implication of* (10) *holds.*

The proof for the reverse implication of (10) *is nearly symmetric.*

iii) and iv) now follow from (10). ∎

Notice, however, that even if we restrict ourselves to deterministic programs, then there are still programs which are protected against $\mathbf{SS}_{\mathrm{T}}$-attacks which may have **SS**-attacks. The program presented in the proof of Theorem III.2 is an example of such a deterministic program.

Corollary IV.5. **SS**-NI $\subsetneq$ $\mathbf{SS}_{\mathrm{T}}$-NI.

Proof: Follows from Theorem III.2. ∎

However, the construction of ω'_j in the proof of Theorem IV.4, together with (9), gives us that the space of attacks to consider when securing deterministic interactive programs is a small one; namely those with stream strategies supplying a finite number of inputs on each input channel.

V. Compositionality

A common scenario for interactive programs is when they interact with other interactive programs. It is therefore of key importance to ensure that the interaction of secure interactive programs does not create an information leak. As we saw in the introduction, a seemingly innocuous leak through message presence in one program can be magnified when that program is run in parallel with other interactive programs. We show that **Strat**-NI is compositional. That is, the parallel composition of noninterfering programs yields a noninterfering program. We thus guarantee the absence of high-bandwidth leaks through message presence in the concurrent setting.

$$\frac{s_1 \xrightarrow{a} s'_1}{s_1 \parallel s_2 \xrightarrow{a} s'_1 \parallel s_2} \qquad \frac{s_2 \xrightarrow{a} s'_2}{s_1 \parallel s_2 \xrightarrow{a} s_1 \parallel s'_2}$$

Figure 2. Labeled Reduction Relation for Interactive Programs in Parallel

In theory and practice, there are scenarios and primitives for almost any wirings of output channels to input channels, fixed or runtime-changing, scoped or unscoped. To stay as general as possible, we leave it to the environment to decide how to route messages. This yields the very simple semantics displayed in Figure 2. Here, any message outputted by resp. inputted to the parallel composition $s_1 \parallel s_2$ of s_1 and s_2 is produced resp. consumed by exactly one of the parallel components s_1 and s_2. A parallel composition of interactive programs is then itself an interactive program, and thus, all the results from Sections II through IV apply to them. This "strategies as glue" approach has much appeal for our security purposes; since all plugging and routing is performed by strategies, and since strategies are general, we can, by picking the right strategy, model different types of composition. For instance, (1), with H_0, H_1, H and L aliased H_0, H_1, M and L, would in practice ideally scope channels H_0 and H_1 to be internal to the parallel composition, and wire output-H_j to input-H_j, leaving input channel M and output channel L as the external interface. A strategy can achieve this wiring by implementing a buffer on H_0 and H_1. A strategy for α which implements a message buffer[2] for channel α is given below, defined using pattern-matching syntax similar to that of Haskell and ML.

$$\begin{array}{lll} \omega_\alpha\ (t.\alpha!v.t'.\alpha?v.t'') & \mid\ (t\restriction_\alpha = t'\restriction_?\restriction_\alpha = \epsilon) & = \omega_\alpha\ t'.t'' \\ \omega_\alpha\ (t.\alpha!v.t') & \mid\ (t\restriction_\alpha = t'\restriction_?\restriction_\alpha = \epsilon) & = v \\ \omega_\alpha\ t & & = \bot \end{array}$$

Using the above definition for H_0 and H_1, we obtain the desired wiring, where no externals influence the communication on H_0 and H_1 (the reason for scoping them).

Note that our compositionality result can only be used to reason about the security of programs which *are* running in parallel. Consider $s_{\mathrm{A}}[s_{\mathrm{B}}]$: a program s_{A} which after some computation steps forks s_{B} as a new thread. The behavior of $s_{\mathrm{A}}[s_{\mathrm{B}}]$ can be given as an IOLTS in terms of parallel composition. When the forking occurs, q becomes a parallel component. Thus, a secure $s_{\mathrm{A}}[s_{\mathrm{B}}]$, in parallel with any IOLTS, yields a secure composition. However, s_{B} is not a parallel component until $s_{\mathrm{A}}[s_{\mathrm{B}}]$ has forked s_{B}. Indeed, if $s_{\mathrm{A}}[s_{\mathrm{B}}] \xrightarrow{t} s'_{\mathrm{A}} \parallel s_{\mathrm{B}}$, then even if s'_{A} and s_{B}, and thus $s'_{\mathrm{A}} \parallel s_{\mathrm{B}}$, are secure, then this does not imply the security of $s_{\mathrm{A}}[s_{\mathrm{B}}]$ as

[2]Inputting from an empty buffer is impossible, and messages in the buffer are inputted in FIFO order.

the occurrence of t can leak information. We discuss how to track information flows in the presence of a forking construct in Section VI.

We now prove our compositionality result. The idea here is that given an attack on $s_{\mathtt{A}} \parallel s_{\mathtt{B}}$, we obtain an attack on either $s_{\mathtt{A}}$ or $s_{\mathtt{B}}$ by incorporating $s_{\mathtt{A}}$ into the attack environment to produce an attack on $s_{\mathtt{B}}$, and vice versa.

Theorem V.1. *For all $s_{\mathtt{A}}$ and $s_{\mathtt{B}}$,*

$$s_{\mathtt{A}}, s_{\mathtt{B}} \in \mathbf{Strat}\text{-NI} \implies s_{\mathtt{A}} \parallel s_{\mathtt{B}} \in \mathbf{Strat}\text{-NI}.$$

Proof: We show the contrapositive. That is,

$$s_{\mathtt{A}} \parallel s_{\mathtt{B}} \notin \mathbf{Strat}\text{-NI} \implies s_{\mathtt{A}} \notin \mathbf{Strat}\text{-NI} \vee s_{\mathtt{B}} \notin \mathbf{Strat}\text{-NI}.$$

Assume $s_{\mathtt{A}} \parallel s_{\mathtt{B}} \notin \mathbf{Strat}$-NI. $s_{\mathtt{A}} \parallel s_{\mathtt{B}} \notin \mathbf{DS}$-NI by Theorem III.5. Then there is a $\mathbf{DS}$-attack $(l, \omega_1, \omega_2, t_1)$ on $s_{\mathtt{A}} \parallel s_{\mathtt{B}}$. Particularly,

$$\forall t_2 \,.\, \omega_2 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t_2} \implies t_2 \not\approx_l t_1. \tag{11}$$

By Lemma III.6,

$$\forall \alpha; \gamma(\alpha) = l_1^{l_2} \,.\, l_2 \not\sqsubseteq l \implies \omega_{2_\alpha} = \emptyset.$$

Assume (towards a contradiction) that $s_{\mathtt{A}}, s_{\mathtt{B}} \in \mathbf{Strat}$-NI. Then, by Definition II.6, we have for $k \in \{\mathtt{A}, \mathtt{B}\}$,

$$\begin{aligned} &\forall l' \,.\, \forall \omega_{1k}, \omega_{2k} \in \mathbf{Strat} \,.\, \omega_{1k} =_{l'} \omega_{2k} \implies \\ &\forall t_{1k} \,.\, \omega_{1k} \models s_k \xrightarrow{t_{1k}} \implies \\ &\exists t_{2k} \,.\, \omega_{2k} \models s_k \xrightarrow{t_{2k}} \ \wedge\ t_{1k} =_{l'} t_{2k}. \end{aligned} \tag{12}$$

Pick $t_{1\mathtt{A}}$ and $t_{1\mathtt{B}}$ such that $s_{\mathrm{A}} \xrightarrow{t_{1\mathtt{A}}}$, $s_{\mathrm{B}} \xrightarrow{t_{1\mathtt{B}}}$ and $t_{1\mathtt{A}} \|\|_{t_1} t_{1\mathtt{B}}$. Here, for any $\hat{t}$, $\hat{t}_1$ and $\hat{t}_2$, $\hat{t}_1 \|\|_{\hat{t}} \hat{t}_2$ iff $\hat{t}$ is an interleaving of $\hat{t}_1$ and $\hat{t}_2$. Recall that $\omega_1 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t_1}$. We construct $\omega_{1\mathtt{A}}$, $\omega_{2\mathtt{A}}$, $\omega_{1\mathtt{B}}$ and $\omega_{2\mathtt{B}}$ for which $\omega_{1\mathtt{A}} =_l \omega_{2\mathtt{A}}$ and $\omega_{1\mathtt{B}} =_l \omega_{2\mathtt{B}}$ which, by (12), contradict (11). Let $j \in \{1,2\}$, $k, \bar{k} \in \{\mathtt{A}, \mathtt{B}\}$, $k \neq \bar{k}$, and, with $\alpha \stackrel{\text{def}}{=} l_1^{l_2}$,

$$\begin{aligned} \omega_{jk_\alpha}(t) = \{ v \mid \exists t'_1, t'_k, t'_{\bar{k}} \,.\, t =_{l_2} t'_k \wedge t'_k \|\|_{t'_1} t'_{\bar{k}} \\ \wedge\, t'_k.\alpha?v \leq_l t_{1k} \wedge s_k \xrightarrow{t'_k.\alpha?v} \\ \wedge\, t'_{\bar{k}} \leq_l t_{1\bar{k}} \qquad \wedge\, s_{\bar{k}} \xrightarrow{t'_{\bar{k}}} \\ \wedge\, t'_1.\alpha?v \leq_l t_1 \ \ \wedge \omega_j \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v} \}. \end{aligned}$$

We show that ω_{jk} are strategies. Let $t =_{l_2} t'$. Assume $\omega_{jk_\alpha}(t) \neq \emptyset$. Then for some v, $v \in \omega_{jk_\alpha}(t)$. Let t'_1, t'_k and $t'_{\bar{k}}$ be the evidence that $v \in \omega_{jk_\alpha}(t)$ The only condition on t for $v \in \omega_{jk_\alpha}(t)$ to hold is $t =_{l_2} t'_k$. Since $t =_{l_2} t'$, $t =_{l_2} t'_k$ by transitivity. Thus t'_1, t'_k and $t'_{\bar{k}}$ are evidence of $v \in \omega_{jk_\alpha}(t')$. So $v \in \omega_{jk_\alpha}(t') \neq \emptyset$. So $\omega_{jk_\alpha}(t) = \omega_{jk_\alpha}(t')$. Let $t =_{l_1} t'$. Since $l_2 \sqsubseteq l_1$, $t =_{l_2} t'$. Thus $\omega_{jk_\alpha}(t) = \omega_{jk_\alpha}(t')$.

We show that $\omega_{1k} =_l \omega_{2k}$. Let t and $\alpha; \gamma(\alpha) = l_1^{l_2}$ be arbitrary. Case on l.

$l_1 \sqsubseteq l$: *Assume wlg. that $v \in \omega_{1k_\alpha}(t)$. $v \in \omega_{2k_\alpha}(t)$ must be shown. Define p_j such that*

$$\begin{aligned} (\exists t'_1 \,.\, p_j(t, t'_1, \hat{v}) \wedge \omega_j \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?\hat{v}}) \\ \iff \hat{v} \in \omega_{jk_\alpha}(t). \end{aligned}$$

Observe that $p_1 = p_2$. Since $v \in \omega_{jk_\alpha}(t)$, we get for some t'_1,

$$p_1(t, t'_1, v) \wedge \omega_1 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v}.$$

Since $\omega_1 =_l \omega_2$, we get $\omega_2 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v}$. Since $p_2 = p_1$, we get $p_2(t, t'_1, v)$. Thus $v \in \omega_{2k_\alpha}(t)$. Thus we have

$$\forall v \,.\, v \in \omega_{1k_\alpha}(t) \iff v \in \omega_{2k_\alpha}(t).$$

$l_1 \not\sqsubseteq l$ and $l_2 \sqsubseteq l$: *Assume wlg. that $v \in \omega_{1k_\alpha}(t)$. We must show $v' \in \omega_{2k_\alpha}(t)$. Define p_j such that*

$$\left(\begin{array}{l} \exists t'_1, t'_k \,.\, p_j(t, t'_1, t'_k) \wedge t'_k.\alpha?\hat{v} \leq_l t_{1k} \wedge s_k \xrightarrow{t'_k.\alpha?\hat{v}} \\ \wedge t'_1.\alpha?\hat{v} \leq_l t_1 \wedge \omega_j \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?\hat{v}} \end{array} \right) \iff \hat{v} \in \omega_{jk_\alpha}(t).$$

Observe that $p_1 = p_2$. Since $v \in \omega_{jk_\alpha}(t)$, we get for some t'_1 and t'_k,

$$\left(\begin{array}{l} p_1(t, t'_1, t'_k) \wedge t'_k.\alpha?v \leq_l t_{1k} \wedge s_k \xrightarrow{t'_k.\alpha?v} \\ \wedge t'_1.\alpha?v \leq_l t_1 \wedge \omega_1 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v} \end{array} \right)$$

Since $\omega_1 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v}$, we have $s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v}$ and $v \in \omega_{1_\alpha}(t'_1)$. Since $\omega_1 =_l \omega_2$, we get $v' \in \omega_{2_\alpha}(t'_1)$ for some v'. By input neutrality, $s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v'}$. So $\omega_2 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t'_1.\alpha?v'}$. Since $t'_k.\alpha?v =_l t'_k.\alpha?v'$, $t'_k.\alpha?v' \leq_l t_{1k}$. Likewise, since $t'_1.\alpha?v =_l t'_1.\alpha?v'$, $t'_1.\alpha?v' \leq_l t_1$. Since $s_k \xrightarrow{t'_k.\alpha?v}$, we get by input neutrality that $s_k \xrightarrow{t'_k.\alpha?v'}$. Since $p_2 = p_1$, we get $p_2(t, t'_1, t'_k)$. Thus $v' \in \omega_{2k_\alpha}(t)$. Thus we have

$$(\exists v \,.\, v \in \omega_{1k_\alpha}(t)) \iff (\exists v' \,.\, v' \in \omega_{2k_\alpha}(t)).$$

$l_2 \not\sqsubseteq l$: *The condition on t and α in Definition II.5 is vacuously true in this case.*

Since t and α were arbitrary, we get $\omega_{1k} =_l \omega_{2k}$.

We show that $\omega_{1\mathtt{A}} \models s_{\mathtt{A}} \xrightarrow{t_{1\mathtt{A}}}$. We already have $s_{\mathtt{A}} \xrightarrow{t_{1\mathtt{A}}}$, $s_{\mathtt{B}} \xrightarrow{t_{1\mathtt{B}}}$, $t_{1\mathtt{A}} \|\|_{t_1} t_{1\mathtt{B}}$ and $\omega_1 \models s_{\mathtt{A}} \parallel s_{\mathtt{B}} \xrightarrow{t_1}$. We proceed by induction in $n = |t_{1\mathtt{A}} \restriction_?|$.

$n = 0$: *Since $s_{\mathtt{A}} \xrightarrow{t_{1\mathtt{A}}}$ and $t_{1\mathtt{A}}$ has no inputs, $\omega_{1\mathtt{A}} \models s_{\mathtt{A}} \xrightarrow{t_{1\mathtt{A}}}$ holds vacuously.*

$n + 1$, given n: *Assume $\omega_{1\mathtt{A}} \models s_{\mathtt{A}} \xrightarrow{t_{1\mathtt{A}}}$ for $t_{1\mathtt{A}}$ with $|t_{1\mathtt{A}} \restriction_?| = n$; this is our induction hypothesis (IH). For some $t''_{1\mathtt{A}}$ with $|t''_{1\mathtt{A}} \restriction_?| = 0$, $t_{1\mathtt{A}} = t'_{1\mathtt{A}}.\alpha?v.t''_{1\mathtt{A}}$. By (IH) we have $\omega_{1\mathtt{A}} \models s_{\mathtt{A}} \xrightarrow{t'_{1\mathtt{A}}}$. By definition of $t'_{1\mathtt{A}}$, we have*

for some $t'_{1\mathtt{B}}$ and t'_1 for which $t'_{1\mathtt{B}} \leq t_{1\mathtt{B}}$, $t'_1 \leq t_1$, and $t'_{1\mathtt{A}} \mid\mid\mid_{t'_1} t'_{2\mathtt{B}}$ that $\omega_{1\mathtt{A}} \models s_\mathtt{A} \parallel s_\mathtt{B} \xrightarrow{t'_1.\alpha?v}$. By $s_\mathtt{A} \xrightarrow{t_{1\mathtt{A}}}$ and $s_\mathtt{B} \xrightarrow{t_{1\mathtt{B}}}$ we get $s_\mathtt{A} \xrightarrow{t_{1\mathtt{A}}.\alpha?v}$ and $s_\mathtt{B} \xrightarrow{t'_{1\mathtt{B}}}$. Since $\leq \subseteq \leq_l$, we get by definition of $\omega_{1\mathtt{A}}$ that $v \in \omega_{1\mathtt{A}}(t'_{1\mathtt{A}})$. Thus $\omega_{1\mathtt{A}} \models s_\mathtt{A} \xrightarrow{t'_{1\mathtt{A}}.\alpha?v}$. Since $|t''_{1\mathtt{A}} \restriction_?| = 0$ and $s_\mathtt{A} \xrightarrow{t_{1\mathtt{A}}}$, we get $\omega_{1\mathtt{A}} \models s_\mathtt{A} \xrightarrow{t_{1\mathtt{A}}}$.

Likewise (swap As *and* Bs*), $\omega_{1\mathtt{B}} \models s_\mathtt{B} \xrightarrow{t_{1\mathtt{B}}}$.*

Assume that $\omega_{2\mathtt{A}} \models s_\mathtt{A} \xrightarrow{t_{2\mathtt{A}}}$ and $\omega_{2\mathtt{B}} \models s_\mathtt{B} \xrightarrow{t_{2\mathtt{B}}}$ such that $t_{1\mathtt{A}} =_l t_{2\mathtt{A}}$ and $t_{1\mathtt{B}} =_l t_{2\mathtt{B}}$. We show that there then is a t_2 for which $t_2 =_l t_1$ and $\omega_2 \models s_\mathtt{A} \parallel s_\mathtt{B} \xrightarrow{t_2}$, contradicting (11)*. We consider the interesting case where $|t_{2\mathtt{A}} \restriction_?| > 0$ and $|t_{2\mathtt{B}} \restriction_?| > 0$ (if, say, $|t_{2\mathtt{A}} \restriction_?| = 0$, then $s_\mathtt{A} \parallel s_\mathtt{B} \notin \mathbf{DS}\text{-NI} \iff s_\mathtt{B} \notin \mathbf{DS}\text{-NI}$). Let $t_{2\mathtt{A}} = t'_{2\mathtt{A}}.i_\mathtt{A}.t''_{2\mathtt{A}}$ and $t_{2\mathtt{B}} = t'_{2\mathtt{B}}.i_\mathtt{B}.t''_{2\mathtt{B}}$ such that $|t''_{2\mathtt{A}} \restriction_?| = 0$ and $|t''_{2\mathtt{B}} \restriction_?| = 0$. Let $i_\mathtt{A} = \alpha_\mathtt{A}?v_\mathtt{A}$ and $i_\mathtt{B} = \alpha_\mathtt{B}?v_\mathtt{B}$. By definition of $(\omega_{2\mathtt{A}})_{\alpha_\mathtt{A}}$ and $(\omega_{2\mathtt{B}})_{\alpha_\mathtt{B}}$, we have $v_\mathtt{A} \in (\omega_{2\mathtt{A}})_{\alpha_\mathtt{A}}(t'_{2\mathtt{A}})$ and $v_\mathtt{B} \in (\omega_{2\mathtt{B}})_{\alpha_\mathtt{B}}(t'_{2\mathtt{B}})$. Thus for some $t^\mathtt{A}_1$ and $t^\mathtt{B}_1$ for which $t^\mathtt{A}_1.i_\mathtt{A} \leq_l t_1$ and $t^\mathtt{B}_1.i_\mathtt{B} \leq_l t_1$, we have $\omega_2 \models s_\mathtt{A} \parallel s_\mathtt{B} \xrightarrow{t^\mathtt{A}_1.i_\mathtt{A}}$ and $\omega_2 \models s_\mathtt{B} \parallel s_\mathtt{B} \xrightarrow{t^\mathtt{B}_1.i_\mathtt{B}}$. Since $\omega_{2_{\alpha'}} = \emptyset$ for all $\alpha' = l'_1{}^{l'_2}$ with $l'_2 \not\sqsubseteq l$, $t^\mathtt{A}_1.i_\mathtt{A} \restriction_? = t^\mathtt{A}_1.i_\mathtt{A} \restriction_? \restriction_l$ and $t^\mathtt{B}_1.i_\mathtt{B} \restriction_? = t^\mathtt{B}_1.i_\mathtt{B} \restriction_? \restriction_l$. Since $t^\mathtt{A}_1.i_\mathtt{A} \leq_l t_1$ and $t^\mathtt{B}_1.i_\mathtt{B} \leq_l t_1$, then either $t^\mathtt{A}_1.i_\mathtt{A} \leq_l t^\mathtt{B}_1.i_\mathtt{B} \leq_l t_1$ or $t^\mathtt{B}_1.i_\mathtt{B} \leq_l t^\mathtt{A}_1.i_\mathtt{A} \leq_l t_1$. Assume $t^\mathtt{B}_1.i_\mathtt{B} \leq_l t^\mathtt{A}_1.i_\mathtt{A} \leq_l t_1$ wlg.. Then by definition of $t^\mathtt{A}_1$ and $t^\mathtt{B}_1$, $t^\mathtt{A}_1.i_\mathtt{A} \restriction_? \restriction_l = t_1 \restriction_? \restriction_l$ ($i_\mathtt{A}$ is l-equivalent with the last observable input in t_1). Thus $t_1 = t'_1.t''_1$ for some t'_1 and t''_1 for which $t^\mathtt{A}_1.i_\mathtt{A} =_l t'_1$ and $|t''_1 \restriction_? \restriction_l| = 0$. Now, there is some $\hat{t}''_1$ for which $\hat{t}''_1 =_l t''_1$ and $t''_{2\mathtt{A}} \mid\mid\mid_{\hat{t}''_1} t''_{2\mathtt{B}}$. For any such $\hat{t}''_1$, since $\left|\hat{t}''_1 \restriction_?\right| = 0$ and, as established before, $\omega_2 \models s_\mathtt{A} \parallel s_\mathtt{B} \xrightarrow{t^\mathtt{A}_1.i_\mathtt{A}}$, $\omega_2 \models s_\mathtt{A} \parallel s_\mathtt{B} \xrightarrow{t^\mathtt{A}_1.i_\mathtt{A}.\hat{t}''_1}$. But $t^\mathtt{A}_1.i_\mathtt{A}.\hat{t}''_1 =_l t_1$, contradicting* (11)*. So, either*

$$\forall t_{2\mathtt{A}}.\, \omega_{2\mathtt{A}} \models s_\mathtt{A} \xrightarrow{t_{2\mathtt{A}}} \implies t_{2\mathtt{A}} \neq_l t_{1\mathtt{A}}, \text{ or}$$

$$\forall t_{2\mathtt{A}}.\, \omega_{2\mathtt{B}} \models s_\mathtt{B} \xrightarrow{t_{2\mathtt{B}}} \implies t_{2\mathtt{A}} \neq_l t_{1\mathtt{A}}.$$

Thus, either $s_\mathtt{A} \notin \mathbf{Strat}\text{-NI}$ or $s_\mathtt{B} \notin \mathbf{Strat}\text{-NI}$. ∎

It is worth noting that the compositionality result does not condition on whether the interactive programs are deterministic or not. This means that programmers can write deterministic programs, establish that the programs are secure using an enforcement mechanism for deterministic programs, freely compose the programs, and obtain a guarantee that the composition is secure, even though $\parallel$ introduces nondeterministic behavior.

VI. Enforcement

This section presents a small nondeterministic imperative programming language with input and output primitives, and develops a type system that differentiates between the security of the presence of messages and the security of their content. We show that the type system enforces **DS**-NI. In addition, we show how the language (and type system) can be safely extended with top-level parallelism, and establish soundness of the extension. Finally, we illustrate how to track flows in the presence of a `fork` command.

A. Syntax

Assume a standard expression language ranged over by e. The commands form a while language, extended with nondeterministic choice ($c_1 \,[]\, c_2$), input ($\mathtt{in}_\alpha(x)$), and output ($\mathtt{out}_\alpha(e)$).

$$\begin{aligned} c ::= {} & x := e \mid \mathtt{in}_\alpha(x) \mid \mathtt{out}_\alpha(e) \mid c_1; c_2 \mid c_1 \,[]\, c_2 \\ \mid {} & \mathtt{if}\ e\ \mathtt{then}\ c_1\ \mathtt{else}\ c_2 \mid \mathtt{while}\ e\ \mathtt{do}\ c \end{aligned}$$

B. Semantics

Let $\mathbb{V}$ be any set containing 0, and let σ range over variable environments, i.e., maps from variables to values. Following [OCC06] let $\sigma(e)$ denote the evaluation of e in σ. The *configurations* are pairs of variable environments and commands. The semantics of commands is of the form $\langle \sigma_1, c_1 \rangle \xrightarrow{a} \langle \sigma_2, c_2 \rangle$, read the configuration $\langle \sigma_1, c_1 \rangle$ evaluates in one step to the configuration $\langle \sigma_2, c_2 \rangle$ with action a.

The rules of the semantics can be found in Table I. Most rules are entirely standard. Nondeterminism and input is modeled by underspecification. In the former case, either of the rules *nd1* and *nd2* can be chosen to evaluate nondeterministic choice. In the latter case, the strategy the program is run against selects which input transitions are possible, see Section II. Terminal configurations contain special-purpose command `skip`.

The relation in Table I forms an IOLTS in the sense of Section II with $\langle \sigma, c \rangle$ as states and $\xrightarrow{\tau}$ denoted $\rightarrow$. The behavior of c is modeled as an IOLTS by $\langle \langle \rangle, c \rangle$, where $\langle \rangle = \lambda x.\, 0$ (the initial configuration of c).

C. Type system

Let the type environments Γ be maps from variables to security levels, l. For clarity let pc denote the security context, used to track implicit flows, and let τ denote security levels of values.

The expression typing judgments are given as $\Gamma \vdash e : \tau$, where $\Gamma(x) \sqsubseteq \tau$ for each x occurring in e, and the typing judgments for commands have the form $pc, l, \Gamma \vdash_\gamma c : l'$, where the *lexical context* pc is a lower bound of the effects (assignments, inputs and outputs) in c, the *blocking context* l is a lower bound of the actions (inputs and outputs) of c, and the *blocking level* l' an upper bound of the security level of the blocking behavior of c. In addition, the typing judgments are parametrized on a channel labeling γ that ranges over L, M, and H.

The type rules for commands are found in Table II. They are standard apart from the addition of the blocking context and the blocking level. Rule *asn* prohibits implicit flows by taking the security context into account, where the security

$$\text{nd1}\ \frac{}{\langle \sigma, c_1 \,[\!]\, c_2\rangle \to \langle \sigma, c_1\rangle} \qquad \text{nd2}\ \frac{}{\langle \sigma, c_1 \,[\!]\, c_2\rangle \to \langle \sigma, c_2\rangle} \qquad \text{asn}\ \frac{}{\langle \sigma, x := e\rangle \to \langle \sigma[x \mapsto \sigma(e)], \texttt{skip}\rangle}$$

$$\text{if1}\ \frac{\sigma(e) \neq 0}{\langle \sigma, \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\rangle \to \langle \sigma, c_1\rangle} \qquad \text{if2}\ \frac{\sigma(e) = 0}{\langle \sigma, \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\rangle \to \langle \sigma, c_2\rangle} \qquad \text{seq1}\ \frac{\langle \sigma_1, c_1\rangle \xrightarrow{a} \langle \sigma_2, c_1'\rangle \quad c_1' \neq \texttt{skip}}{\langle \sigma_1, c_1; c_2\rangle \xrightarrow{a} \langle \sigma_2, c_1'; c_2\rangle}$$

$$\text{wh1}\ \frac{\sigma(e) \neq 0}{\langle \sigma, \texttt{while}\ e\ \texttt{do}\ c\rangle \to \langle \sigma, c; \texttt{while}\ e\ \texttt{do}\ c\rangle} \qquad \text{wh2}\ \frac{\sigma(e) = 0}{\langle \sigma, \texttt{while}\ e\ \texttt{do}\ c\rangle \to \langle \sigma, \texttt{skip}\rangle} \qquad \text{seq2}\ \frac{\langle \sigma_1, c_1\rangle \xrightarrow{a} \langle \sigma_2, \texttt{skip}\rangle}{\langle \sigma_1, c_1; c_2\rangle \xrightarrow{a} \langle \sigma_2, c_2\rangle}$$

$$\text{in}\ \frac{}{\langle \sigma, \texttt{in}_\alpha(x)\rangle \xrightarrow{\alpha?v} \langle \sigma[x \mapsto v], \texttt{skip}\rangle} \qquad \text{out}\ \frac{}{\langle \sigma, \texttt{out}_\alpha(e)\rangle \xrightarrow{\alpha!\sigma(e)} \langle \sigma, \texttt{skip}\rangle}$$

Table I
SEMANTICS OF COMMANDS

$$\text{nd}\ \frac{pc, l_1, \Gamma \vdash_\gamma c_1 : l_2 \quad pc, l_1, \Gamma \vdash_\gamma c_2 : l_2}{pc, l_1, \Gamma \vdash_\gamma c_1 \,[\!]\, c_2 : l_2} \qquad \text{asn}\ \frac{\Gamma \vdash e : \tau \quad pc \sqcup \tau \sqsubseteq \Gamma(x)}{pc, l, \Gamma \vdash_\gamma x := e : L} \qquad \text{seq}\ \frac{pc, l_1, \Gamma \vdash_\gamma c_1 : l_2 \quad pc, l_1 \sqcup l_2, \Gamma \vdash_\gamma c_2 : l_3}{pc, l_1, \Gamma \vdash_\gamma c_1; c_2 : l_2 \sqcup l_3}$$

$$\text{inL}\ \frac{\gamma(\alpha) = \mathsf{L}}{L, L, \Gamma \vdash_\gamma \texttt{in}_\alpha(x) : L} \qquad \text{inM}\ \frac{\gamma(\alpha) = \mathsf{M} \quad H \sqsubseteq \Gamma(x)}{L, L, \Gamma \vdash_\gamma \texttt{in}_\alpha(x) : L} \qquad \text{inH}\ \frac{\gamma(\alpha) = \mathsf{H} \quad H \sqsubseteq \Gamma(x)}{pc, l_1, \Gamma \vdash_\gamma \texttt{in}_\alpha(x) : l_2}$$

$$\text{outL}\ \frac{\gamma(\alpha) = \mathsf{L} \quad \Gamma \vdash e : L}{L, L, \Gamma \vdash_\gamma \texttt{out}_\alpha(e) : L} \qquad \text{outM}\ \frac{\gamma(\alpha) = \mathsf{M}}{L, L, \Gamma \vdash_\gamma \texttt{out}_\alpha(e) : L} \qquad \text{outH}\ \frac{\gamma(\alpha) = \mathsf{H}}{pc, l, \Gamma \vdash_\gamma \texttt{out}_\alpha(e) : L}$$

$$\text{if}\ \frac{\Gamma_1 \vdash e : \tau \quad pc \sqcup \tau, l_1, \Gamma \vdash_\gamma c_1 : l_2 \quad pc \sqcup \tau, l_1, \Gamma \vdash_\gamma c_2 : l_2}{pc, l_1, \Gamma \vdash_\gamma \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2 : l_2} \qquad \text{wh}\ \frac{\Gamma \vdash e : \tau \quad pc \sqcup \tau, l_1, \Gamma \vdash_\gamma c : l_2 \quad l_2 \sqsubseteq l_1}{pc, l_1, \Gamma \vdash_\gamma \texttt{while}\ e\ \texttt{do}\ c : pc \sqcup \tau \sqcup l_2}$$

$$\text{sb}\ \frac{pc_2, l_2, \Gamma \vdash_\gamma c : l_3 \quad pc_1 \sqsubseteq pc_2 \quad l_1 \sqsubseteq l_2 \quad l_3 \sqsubseteq l_4}{pc_1, l_1, \Gamma \vdash_\gamma c : l_4}$$

Table II
TYPE RULES OF COMMANDS

context is raised to the security level of the expressions guarding the control flow in rules *if*, and *wh*.

The blocking level expresses whether the blocking behavior of a command depends on secrets or not. Hence, a command with secret blocking level, $pc, l, \Gamma \vdash_\gamma c : H$, may diverge and/or input-block, depending on secrets. This implies that any succeeding commands cannot be allowed to have public actions, i.e., the succeeding command must be typable in a secret blocking context, as seen in rule *seq*. Otherwise a leak occurs, as demonstrated in the introduction. There the example program $\texttt{in}_H(x); \texttt{out}_L(1)$ was used, but any source of secret blocking must lead to the same restrictions, as illustrated by rules *inH* and *wh*.

In contrast to previous work [OCC06], our type system allows secret blocking levels. A program well-typed with a secret blocking level can block depending on secret information, but that blocking will not influence public actions performed by the program. This discipline is similar to the handling of while loops by Boudol and Castellani [BC02] and Smith [Smi01], where assignments to public variables are prevented from happening after any possibility of entering loops with high guards. A key difference in our approach is the blocking context, which distinguishes internal side effects (due to assignment) from external side effects (due to output and input, which are not treated by Boudol, Castellani, or Smith). Assuming h is secret and l is public, it allows us to rightfully accept secure programs like $\texttt{in}_H(x); l := 1$ and $(\texttt{while}\ h\ \texttt{do}\ h := h); l := 1$, which are problematic in the shared-memory concurrency setting considered by Boudol, Castellani, and Smith.

All commands with public actions — rules *inL*, *inM*, *outL*, and *outM* — are constrained to run in public security and blocking level. This means that the program $\texttt{in}_H(x); \texttt{out}_L(1)$ is not accepted by the type system, whereas $\texttt{in}_M(x); \texttt{out}_L(1)$ is. Also, the order of instructions matters as indicated by the threading in rule *seq*. Hence, $\texttt{out}_L(1); \texttt{in}_H(x)$ is accepted by the type system since the secret-presence input occurs after the public output.

Theorem VI.1. *Soundness of the type system.*

$$pc, l_1, \Gamma \vdash_\gamma c : l_2 \implies \langle\langle\rangle, c\rangle \in \mathbf{DS}\text{-NI}$$

Proof: The proof can be found in the appendix. ∎

D. Parallel Composition

We add top-level parallel composition to the language. A program p is a command or two programs in parallel.

$$p ::= c \mid p_1 \parallel p_2$$

We obtain a semantics for parallel composition by lifting the parallel composition operator to the level of IOLTSs; with

$$[\![c]\!] = \langle\langle\rangle, c\rangle \qquad [\![p_1 \parallel p_2]\!] = [\![p_1]\!] \parallel [\![p_2]\!],$$

the semantics for parallel composition is as presented in Section V. Typing parallel composition is done by typing all participating commands using the same channel labeling.

$$\text{parc}\frac{pc, l_1, \Gamma \vdash_\gamma c : l_2}{\vdash_\gamma c} \qquad \text{par}\frac{\vdash_\gamma p_1 \qquad \vdash_\gamma p_2}{\vdash_\gamma p_1 \parallel p_2}$$

Since the type system guarantees $\mathbf{DS}$-NI, the soundness of parallel composition follows from the soundness of the type system and the compositionality result of Section V.

Theorem VI.2. *Soundness of parallel composition.*

$$\vdash_\gamma p \Longrightarrow [\![p]\!] \in \mathbf{DS}\text{-NI}$$

Proof: Immediate from Theorems VI.1 and V.1. ■

E. Fork command

On a last note, say our language contains a $\texttt{fork}(c)$ primitive, which, when executed, will cause $\langle \sigma, c \rangle$ to run in parallel with the executing IOLTS, where σ is either $\langle\rangle$ or (a copy of) the variable environment of the executing IOLTS. If c produces L effects, and $\texttt{fork}(c)$ is executed in a H context, then an information leak can occur, as in $\texttt{in}_\mathsf{M}(h); \texttt{if } h \texttt{ then fork}(\texttt{out}_\mathsf{L}(0)) \texttt{ else } h := h$. To track flows in the presence of $\texttt{fork}(c)$, we suggest typing c under the context of creation of $\langle \sigma, c \rangle$, as in the following rule.

$$\text{fork}\frac{pc, l, \Gamma \vdash_\gamma c : l'}{pc, l, \Gamma \vdash_\gamma \texttt{fork}(c) : l}$$

Note the l' because c cannot, by blocking, constrain the behavior of the IOLTS executing $\texttt{fork}(c)$.

VII. Related work

Security of interactive systems has been investigated in the context of process calculi [FG95], [RS99], [HVY00], [Rya01], [HY02], [Pot02], [Kob03] and event-based abstractions [Man00], [Man01], [SM02]. Connections with security models for more concrete programming languages have been made [MS03], [FRS05]. However, relatively little has been done on tracking the flow of information through language constructs in interactive languages.

Strategy-based models: Wittbold and Johnson [WJ90] are the first to define strategy-based information-flow security. In a language-based setting, O'Neill et al. [OCC06] investigate the security of interactive programs in the presence of user strategies. They present a strategy-based security condition and a type system that guarantees security. Our framework generalizes this work by distinguishing the security level for message presence and removing the assumption of the totality for strategies. Compared to the type system by O'Neill et al., our type system (i) tracks the level of message presence, (ii) handles parallel composition, and (iii) has more permissive rules for loops.

Clark and Hunt [CH08] prove that it makes no difference in a deterministic setting whether the environment is represented by strategies or streams. Our results can be seen as a generalization along two dimensions. The first dimension allows both total and nontotal strategies. The second dimension parametrizes in the presence level for channels.

Stream-based models: Streams are commonly used for representing the interaction environment of programs. Our generalization of Clark and Hunt's results ensures that using streams does not sacrifice generality, as long as programs are deterministic.

Sabelfeld and Mantel [SM02] investigate the impact of different types of channels (secret, encrypted, public) and different types of communication (synchronous and asynchronous) on information-flow security. The encrypted channel is similar to our low-presence channel, where only the presence (not the content) of messages is visible to attackers. The origins of presence and content levels are in security labels for datatypes. For example, Jif [Mye99], [MZZ$^+$01] allows arrays, where the length of the array is public but the individual elements are secret.

Communication is modeled by streams in security formalizations by Askarov et al. [AHS08] for a language with cryptographic primitives, and by Askarov and Sabelfeld [AS09] for a language with dynamic code evaluation and declassification primitives.

Askarov et al. [AHSS08] clarify the impact of leaking information via intermediate output. They investigate a condition that is insensitive to computation progress and show that the attacker cannot learn secret information in polynomial time in the size of the secret. This implies that restrictions on language constructs that might result in abnormal termination or divergence, originating in classical security analysis [DD77], [VSI96] and supported in modern information-flow tools Jif [MZZ$^+$01], FlowCaml [Sim03], and the SPARK Examiner [Bar03], [CH04], are not strong enough to prevent brute-force attacks.

Bohannon et al. [BPS$^+$09] propose security definitions for reactive systems that correspond to four indistinguishability relations on streams. They emphasize CP-security (sensitive to computation progress) and ID-security (insensitive to computation progress and thus similar to the one by Askarov et al. [AHSS08]).

In a stream-based setting, Rafnsson and Sabelfeld [RS11] distinguish the security level of message presence and content, accommodate new handler creation, and deploy output buffering to reduce leaks through intermediate output to at most one bit per consumed public input.

Devriese and Piessens [DP10] suggest splitting the execution of a program onto threads operating at different security levels. Only the thread at a given level is allowed to produce output on a channel labeled with the level. An input at a given security level is processed by the thread at that level and forwarded to threads that are above in the security hierarchy. With some care taken when scheduling the threads (as spelled out by Kashyap et al. [KWH11]), it

is possible to achieve both timing- and termination-sensitive noninterference.

Local interaction: Almeida Matos et al. [ABC07] consider local synchronous composition of threads under cooperative scheduling. They study a reactive setting, where treads can broadcast and react to local signals. They propose a formalization of noninterference for this setting and a type system that enforces it. The focus is primarily on suspension features and leaks associated with them.

VIII. Conclusion

We have presented a generalized framework for securing interactive programs. The framework drops the assumption from previous work that strategies must be always able to feed new input into the system. Further, the framework enables fine-grained security types for channels, distinguishing between the security level of message presence and content.

We have established compositionality of the security condition: assorted compositions of secure threads result in a secure thread pool. We have showed an enforcement of the condition via a type system. The type system capitalizes on the distinction between the security level of message presence and content, as well as on the compositionality properties.

Future work is focused on exploring the impact of nondeterminism on the security condition. We are interested in tight stream-based approximations of strategy-based security as well as in a type system that tracks the interplay between nondeterminism and interaction.

Acknowledgments

This work was funded by the European Community under the WebSand project and the Swedish research agencies SSF and VR.

References

[ABC07] A. Almeida Matos, G. Boudol, and I. Castellani. Typing noninterference for reactive programs. *Journal of Logic and Algebraic Programming*, 72:124–156, 2007.

[AHS08] A. Askarov, D. Hedin, and A. Sabelfeld. Cryptographically-masked flows. *Theoretical Computer Science*, 402:82–101, August 2008.

[AHSS08] A. Askarov, S. Hunt, A. Sabelfeld, and D. Sands. Termination-insensitive noninterference leaks more than just a bit. In *Proc. European Symp. on Research in Computer Security*, volume 5283 of *LNCS*, pages 333–348. Springer-Verlag, October 2008.

[AS09] A. Askarov and A. Sabelfeld. Tight enforcement of information-release policies for dynamic languages. In *Proc. IEEE Computer Security Foundations Symposium*, July 2009.

[Bar03] J. Barnes. *High Integrity Software: The SPARK Approach to Safety and Security*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[BC02] G. Boudol and I. Castellani. Non-interference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1):109–130, June 2002.

[BPS+09] Aaron Bohannon, Benjamin C. Pierce, Vilhelm Sjöberg, Stephanie Weirich, and Steve Zdancewic. Reactive noninterference. In *ACM Conference on Computer and Communications Security*, pages 79–90, November 2009.

[CH04] R. Chapman and A. Hilton. Enforcing security and safety models with an information flow analysis tool. *ACM SIGAda Ada Letters*, 24(4):39–46, 2004.

[CH08] D. Clark and S. Hunt. Noninterference for deterministic interactive programs. In *Workshop on Formal Aspects in Security and Trust (FAST'08)*, October 2008.

[DD77] D. E. Denning and P. J. Denning. Certification of programs for secure information flow. *Comm. of the ACM*, 20(7):504–513, July 1977.

[DP10] D. Devriese and F. Piessens. Non-interference through secure multi-execution. In *Proc. IEEE Symp. on Security and Privacy*, May 2010.

[FG95] R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *J. Computer Security*, 3(1):5–33, 1995.

[FRS05] R. Focardi, S. Rossi, and A. Sabelfeld. Bridging language-based and process calculi security. In *Proc. Foundations of Software Science and Computation Structure*, volume 3441 of *LNCS*, pages 299–315. Springer-Verlag, April 2005.

[HVY00] K. Honda, V. Vasconcelos, and N. Yoshida. Secure information flow as typed process behaviour. In *Proc. European Symp. on Programming*, volume 1782 of *LNCS*, pages 180–199. Springer-Verlag, 2000.

[HY02] K. Honda and N. Yoshida. A uniform type structure for secure information flow. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 81–92, January 2002.

[Kob03] N. Kobayashi. Type-based information flow analysis for the pi-calculus. Technical Report TR03-0007, Tokyo Institute of Technology, October 2003.

[KWH11] V. Kashyap, B. Wiedermann, and B. Hardekopf. Timing- and termination-sensitive secure information flow: Exploring a new approach. In *Proc. IEEE Symp. on Security and Privacy*, 2011.

[Man00] H. Mantel. Possibilistic definitions of security – An assembly kit –. In *Proc. IEEE Computer Security Foundations Workshop*, pages 185–199, July 2000.

[Man01] H. Mantel. Information flow control and applications—Bridging a gap. In *Proc. Formal Methods Europe*, volume 2021 of *LNCS*, pages 153–172. Springer-Verlag, March 2001.

[McC87] D. McCullough. Specifications for multi-level security and hook-up property. In *Proc. IEEE Symp. on Security and Privacy*, pages 161–166, April 1987.

[MS03] H. Mantel and A. Sabelfeld. A unifying approach to the security of distributed and multi-threaded programs. *J. Computer Security*, 11(4):615–676, September 2003.

[Mye99] A. C. Myers. JFlow: Practical mostly-static information flow control. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 228–241, January 1999.

[MZZ+01] A. C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. Jif: Java information flow. Software release. Located at http://www.cs.cornell.edu/jif, July 2001.

[OCC06] K. O'Neill, M. Clarkson, and S. Chong. Information-flow security for interactive programs. In *Proc. IEEE Computer Security Foundations Workshop*, pages 190–201, July 2006.

[Pot02] F. Pottier. A simple view of type-secure information flow in the pi-calculus. In *Proc. IEEE Computer Security Foundations Workshop*, pages 320–330, June 2002.

[RS99] P. Ryan and S. Schneider. Process algebra and non-interference. In *Proc. IEEE Computer Security Foundations Workshop*, pages 214–227, June 1999.

[RS11] W. Rafnsson and A. Sabelfeld. Limiting information leakage in event-based communication. In *Proc. ACM Workshop on Programming Languages and Analysis for Security (PLAS)*, June 2011.

[Rya01] P. Ryan. Mathematical models of computer security—tutorial lectures. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *LNCS*, pages 1–62. Springer-Verlag, 2001.

[Sim03] V. Simonet. The Flow Caml system. Software release. Located at http://cristal.inria.fr/~simonet/soft/flowcaml, July 2003.

[SM02] A. Sabelfeld and H. Mantel. Static confidentiality enforcement for distributed programs. In *Proc. Symp. on Static Analysis*, volume 2477 of *LNCS*, pages 376–394. Springer-Verlag, September 2002.

[Smi01] G. Smith. A new type system for secure information flow. In *Proc. IEEE Computer Security Foundations Workshop*, pages 115–125, June 2001.

[VSI96] D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *J. Computer Security*, 4(3):167–187, 1996.

[WJ90] J. T. Wittbold and D. M. Johnson. Information flow in nondeterministic systems. In *Proc. IEEE Symp. on Security and Privacy*, pages 144–161, 1990.

APPENDIX

A. Notation

In the following, $\$ ::= ? \mid !$. The inputs in t, written $t\restriction_?$, is given by

$$\alpha\$v.t'\restriction_? = \begin{cases} \alpha\$v.(t'\restriction_?) & \text{, if } \$ = ? \\ t\restriction_? & \text{, otherwise.} \end{cases}$$

The outputs in t, written $t\restriction_!$, is defined analogously. The α-messages in t, written $t\restriction_\alpha$, is given by

$$\alpha\$v.t'\restriction'_\alpha = \begin{cases} \alpha\$v.(t'\restriction_\alpha) & \text{, if } \alpha' = \alpha \\ t\restriction_\alpha & \text{, otherwise.} \end{cases}$$

The l-observables in t, written $t\restriction_l$, is given by

$$\alpha\$v.t\restriction_l = \begin{cases} \alpha\$v.(t\restriction_l) & \text{, if } \gamma(\alpha) = l_1^{l_2} \text{ and } l_1 \sqsubseteq l \\ \alpha\$\,\square\,.(t\restriction_l) & \text{, if } \gamma(\alpha) = l_1^{l_2} \text{ and } l_2 \sqsubseteq l \\ t\restriction_l & \text{, otherwise.} \end{cases}$$

For all of these $\restriction$-operators, $\epsilon\restriction = \epsilon$ and $(\tau.t)\restriction = t\restriction$. $\mathcal{P}(A)$ is the powerset of A. $|A|$ is the number of elements of A. $|t|$ is the number of actions in t. $t' \leq t$ if there is some t' for which $t = t't''$.

B. Bounded Strategies

Consider strategies which, for each α, are total on the set of traces traces which contains a number of α-inputs less than some bound n. We refer to these as bounded strategies.

Definition A.1. ω is *bounded* if for all α, there is an n for which $|t\restriction_\alpha\restriction_?| < n \iff \omega_\alpha(t) \neq \emptyset$.

Let W_{B} denote the set of bounded strategies in W.

Definition A.2. ω *refines* ω', written $\omega \leq \omega'$, iff for all α and t, $\omega_\alpha(t) \subseteq \omega'_\alpha(t)$.

Lemma A.3 ([CH08])**.** *If $\omega \leq \omega'$ and $\omega \models s \xrightarrow{t}$, then $\omega' \models s \xrightarrow{t}$.*

It turns out we only need to consider attacks containing deterministic, bounded strategies when checking for insecure flows in an interactive program.

Proposition A.4. $\mathbf{Strat}\text{-NI} = \mathbf{DS}_{\mathrm{B}}\text{-NI}$.

Proof: We prove $\mathbf{DS}\text{-NI} = \mathbf{DS}_{\mathrm{B}}\text{-NI}$; the result will then follow from Theorem III.5. $\mathbf{DS}_{\mathrm{B}} \subseteq \mathbf{DS}$ by definition of $\mathbf{Strat}_{\mathrm{B}}$. By Lemma II.8, $\mathbf{DS}\text{-NI} \subseteq \mathbf{DS}_{\mathrm{B}}\text{-NI}$. We now show $\mathbf{DS}\text{-NI} \supseteq \mathbf{DS}_{\mathrm{B}}\text{-NI}$. That is,

$$\forall s.\, s \in \mathbf{DS}_{\mathrm{B}}\text{-NI} \implies s \in \mathbf{DS}\text{-NI}.$$

We show instead the contrapositive. That is,

$$\forall s.\, s \notin \mathbf{DS}\text{-NI} \implies s \notin \mathbf{DS}_{\mathrm{B}}\text{-NI}. \tag{13}$$

Let $s \notin \mathbf{DS}\text{-NI}$ be given. Then there is a $\mathbf{DS}$-attack $(l, \omega_1, \omega_2, t_1)$ on s. Particularly,

1) $\omega_1 =_l \omega_2$,
2) $\omega_1 \models s \xrightarrow{t_1}$,
3) $\forall t_2.\, \omega_2 \models s \xrightarrow{t_2} \implies t_2 \neq_l t_1$.

Let

$$\omega'_{j_\alpha}(t) = \begin{cases} \bot & \text{, if } |t\restriction_\alpha\restriction_?| \geq |t_1\restriction_\alpha\restriction_?| \\ \omega_{j_\alpha}(t) & \text{, otherwise.} \end{cases}$$

We must show that

i) $\omega'_1 =_l \omega'_2$,
ii) ω'_j is a strategy,
iii) $\omega'_1 \models s \xrightarrow{t_1}$,
iv) $\forall t_2.\, \omega'_2 \models s \xrightarrow{t_2} \implies t_1 \neq_l t_2$.

We have i) from 1) since, for all t and α,

$$\omega_{1_\alpha}(t) \neq \omega'_{1_\alpha}(t) = \bot \implies \omega'_{2_\alpha}(t) = \bot$$
$$\omega_{2_\alpha}(t) \neq \omega'_{2_\alpha}(t) = \bot \implies \omega'_{1_\alpha}(t) = \bot.$$

For α for which $\gamma(\alpha) = l_1^{l_2}$, since $t =_{l_2} t' \implies |t\restriction_\alpha\restriction_?| = |t'\restriction_\alpha\restriction_?|$ and $t =_{l_1} t' \implies t =_{l_2} t'$, then either $\omega_{j_\alpha}(t) = \omega_{j_\alpha}(t') = \omega'_{j_\alpha}(t) = \omega'_{j_\alpha}(t') = \bot$ or $\omega'_{j_\alpha}(t) = \omega_{j_\alpha}(t)$ and $\omega'_{j_\alpha}(t') = \omega_{j_\alpha}(t')$. Thus, since ω_j are strategies, ii) holds. Since for all $\alpha?v$ and t, $t.\alpha?v \leq t_1 \implies |t\restriction_\alpha\restriction_?| < |t_1\restriction_\alpha\restriction_?|$, we get $\omega'_{1_\alpha}(t) = \omega_{1_\alpha}(t) = v$. Thus $\omega_1 \models t_1$. Since $s \xrightarrow{t_1}$, we get iii) by definition of $s \xrightarrow{t_1}$. We have iv) by Lemma A.3 since $\omega'_2 \leq \omega_2$.

By Lemmas III.7 and III.8, $\forall t, \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l$,

$$\omega'_{1_\alpha}(t) = \bot \iff \omega'_{2_\alpha}(t) = \bot \tag{14}$$

and $\forall t, \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l, v$,

$$\omega'_2 \models s \xrightarrow{t} s' \wedge s' \xrightarrow{\alpha?v} \wedge \omega'_{2_\alpha}(t) = \bot \implies t.\alpha?v \not\leq_l t_1, \forall v. \tag{15}$$

In particular, this holds if we fix v to a constant k. Let

$$\hat{\omega}'_{j_\alpha}(t) = \begin{cases} k & \text{, if } |t\restriction_\alpha\restriction_?| < |t_1\restriction_\alpha\restriction_?| \wedge \omega'_{j_\alpha}(t) = \bot, \\ \omega'_{j_\alpha}(t) & \text{, otherwise.} \end{cases}$$

$$\omega''_{1_\alpha}(t) = \hat{\omega}'_{1_\alpha}(t)$$

$$\omega''_{2_\alpha}(t) = \begin{cases} \bot & \text{, if } \alpha = l_1^{l_2} \wedge l_2 \not\sqsubseteq l, \\ \hat{\omega}'_{2_\alpha}(t) & \text{, otherwise.} \end{cases}$$

We must show that

a) $\omega''_1 =_l \omega''_2$,
b) ω''_j *is a strategy,*
c) $\omega''_1 \models s \xrightarrow{t_1}$,
d) $\omega''_2 \models s \xrightarrow{t_2} \implies t_1 \neq_l t_2$, $\forall t_2$.

We have a) from i) since, for all t and $\alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l$,

$$\omega'_{1_\alpha}(t) \neq \omega''_{1_\alpha}(t) = \bot \implies \omega''_{2_\alpha}(t) = \bot$$
$$\omega'_{2_\alpha}(t) \neq \omega''_{2_\alpha}(t) = \bot \implies \omega''_{1_\alpha}(t) = \bot.$$

It is easy to see that $\forall t, \alpha; \gamma(\alpha) = l_1^{l_2}; l_2 \sqsubseteq l$,

$$\omega'_{1_\alpha}(t) = \omega'_{2_\alpha}(t) = \bot \implies \omega''_{1_\alpha}(t) = \omega''_{2_\alpha}(t)$$

This, i) and (14) *gives a). It is easy to see that $\forall t, \alpha$,*

$$\omega'_{1_\alpha}(t) \neq \bot \implies \omega'_{1_\alpha}(t) = \omega''_{j_\alpha}(t).$$

This, and a), gives c). By (15) *and by definition of ω''_{2_α}, d) holds.*

By definition of ω'_j and ω''_j, ω''_1 and ω''_2 are deterministic, and bounded, strategies. Thus $s \notin \mathbf{DS}_\mathrm{B}$-NI. Since s was arbitrary, (13) *holds.* ∎

Lemma III.6 follows from the proof of Proposition A.4.

C. Soundness of the type system

We perform the proof in the instrumented semantics of O'Neill et al. [OCC06] $(c, \sigma, \psi, t, \omega)$ extended with our richer model of channels. This extension is straightforward and does not significantly change the proofs.

In essence our type system is the type system of O'Neill et al. extended with the blocking context and blocking level. For programs typed $pc, L, \Gamma \vdash_\gamma c : L$ their soundness proof applies with minor modifications, since a public blocking context guarantees that the program is free from secret blocking. We begin by establishing a few lemmas relating our type system and semantics to the type system of O'Neill et al.

Lemma A.5. *If channels α s.t. $\gamma(\alpha) = \mathsf{M}$ on the left hand side are interpreted as H on the right hand side we have the following result.*

$$pc, L, \Gamma \vdash_\gamma c : L \implies \Gamma \vdash c : pc\ cmd$$

Proof: By structural induction on c. ∎

Here, $\Gamma \vdash c : pc\ cmd$ is the typing judgment of [OCC06].

Semantically, we have the following correspondence between our semantics and the semantics of [OCC06].

Lemma A.6. *It holds that*

$$\omega \models \langle \sigma, c \rangle \xrightarrow{t} \implies \exists \psi, \psi', c', \sigma' \ .\ (c, \sigma, \psi, \langle\rangle, \omega) \to (c', \sigma', \psi', t, \omega)$$

Proof: The existence of ψ and ψ' corresponding to the nondeterministic choices is immediate. The result follows. ∎

For programs free from blocking the correspondence goes the other direction.

Lemma A.7. *For commands c s.t. $pc, L, \Gamma \vdash_\gamma c : L$ it holds that*

$$(c, \sigma, \psi, \langle\rangle, \omega) \to (c', \sigma', \psi', t, \omega) \implies \omega \models \langle \sigma, c \rangle \xrightarrow{t}$$

Proof: The result follows from the fact that c is free from secret blocking, which is given by $pc, L, \Gamma \vdash_\gamma c : L$. ∎

Theorem A.8. *Soundness of the type system.*

$$pc, l_1, \Gamma \vdash_\gamma c : l_2 \implies \langle\langle\rangle, c\rangle \in \mathbf{DS}\text{-NI}$$

Proof: Let $\sim_L$ be defined as in [OCC06] with the exception that $\omega_1 \sim_L \omega_2$ is taken to be $\omega_1 =_L \omega_2$. The proof proceeds by case analysis on $pc, l_1, \Gamma \vdash_\gamma c : l_2$.

$pc, L, \Gamma \vdash_\gamma c : L$ This case is equivalent to the proof of [OCC06]. Given $\omega_1 =_L \omega_2$, and $\sigma_1 \sim_L \sigma_2$, for $\omega_1 \models \langle \sigma_1, c \rangle \xrightarrow{t_1}$ show that $\omega_2 \models \langle \sigma_2, c \rangle \xrightarrow{t_2}$ s.t. $t_1 \sim_L t_2$ exists. Now, Lemma A.6 gives us that there exists $\psi_1, \psi'_1, c'_1, \sigma'_1$, s.t. $(c, \sigma_1, \psi_1, \langle\rangle, \omega_1) \to (c'_1, \sigma'_1, \psi'_1, t_1, \omega_1)$, and Lemma A.5 gives us that $\Gamma \vdash c : pc\ cmd$. Now, Theorem 2 of [OCC06] gives us that there exists there exists $\psi_2, \psi'_2, c'_2, \sigma'_2$, s.t. $(c, \sigma_2, \psi_2, \langle\rangle, \omega_2) \to (c'_2, \sigma'_2, \psi'_2, t_2, \omega_2)$ and $t_1 \sim_L t_2$. Now, Lemma A.7 allows us to establish $\omega_2 \models \langle \sigma_2, c \rangle \xrightarrow{t_2}$ and the result follows.

$pc, H, \Gamma \vdash_\gamma c : l$ Immediate, since c is low-silent.

$pc, L, \Gamma \vdash_\gamma c : H$ Assume $\omega_1 =_L \omega_2$, and $\sigma_1 \sim_L \sigma_2$, and two executions $\omega_1 \models \langle \sigma_1, c \rangle \xrightarrow{t_1}$, and $\omega_2 \models \langle \sigma_2, c \rangle \xrightarrow{t_2}$. It is easy to show that there exists a prefix c', and two suffixes c_1, c_2 such that $\omega_1 \models \langle \sigma_1, c'; c_1 \rangle \xrightarrow{t_1}$, and $\omega_2 \models \langle \sigma_2, c'; c_2 \rangle \xrightarrow{t_2}$, where $pc, L, \Gamma \vdash_\gamma c' : L$, $pc, L, \Gamma \vdash_\gamma c_1 : H$, and $pc, L, \Gamma \vdash_\gamma c_2 : H$, and where c_1, c_2 is prefixed by either $\mathtt{out}_H(x)$ a secret conditional or a secret while. In either case, we have that c_1 and c_2 are low-silent (the secret conditional, or secret while are low-silent in the bodies, since they constitute secret contexts; any suffixes typed $pc, H, \Gamma \vdash_\gamma c : H$ and are, hence, low-silent. Now, the result for $pc, L, \Gamma \vdash_\gamma c' : L$ from above allows us to establish low-equivalence on the parts of the traces leading up to c_1 and c_2; from this $t_1 \sim_L t_2$ follows.

∎