Information Flow Security in Boundary Ambients ⋆

Chiara Braghin [a,*], Agostino Cortesi [b], Riccardo Focardi [b]

[a] *Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, via Bramante 65, Crema, Italy*

[b] *Dipartimento di Informatica, Università Ca' Foscari di Venezia, via Torino 155, Mestre, Italy*

Abstract

A variant of the Mobile Ambient calculus, called *Boundary Ambients*, is introduced, supporting the modelling of multi-level security policies. Ambients that may guarantee to properly protect their content are explicitly identified as *boundaries*: a boundary can be seen as a resource access manager for confidential data. In this setting, absence of *direct* information leakage is granted as soon as the initial process satisfies some syntactic conditions. We then give a new notion of non-interference for Boundary Ambients aiming at capturing *indirect* flows, too. We design a Control Flow Analysis that computes an over-approximation of all ambients that may be affected at run-time by high-level data and we show that this static analysis can be used to enforce non-interference, i.e., to statically detect that no (direct or indirect) information leakage is ever possible at run-time.

1 Introduction

The Mobile Ambient calculus [12] is a very interesting workbench to reason about mobility related issues, in which security plays a crucial role. Our starting point is the "core" version of Mobile Ambients, where no communication primitives are present and the only possible actions are represented by the

⋆ This work is an extended and revised version of [6] and has been partially supported by the FIRB project RBAU018RCZ 002.

* Corresponding author.

Email addresses: braghin@dti.unimi.it (Chiara Braghin), cortesi@dsi.unive.it (Agostino Cortesi), focardi@dsi.unive.it (Riccardo Focardi).

moves performed by mobile processes. We choose this basic calculus because it allows us to study a very general notion of information flow security which should be easily scalable also to more refined versions of the calculus such as, e.g., Boxed Ambients [10], Safe Ambients [26], or BioAmbients [33].

Among the security models, the Bell-LaPadula model (BLP) [2] provides a framework for handling data of different clearance levels (*high* and *low*, for simplicity), and for this reason is also called *multi-level security* model. The purpose of the model is to confine sensitive data at its correct level by means of two access rules that are imposed by the system: *No Read up* prevents users from accessing information for which they are not cleared to access; *No Write down* prevents users (or more importantly software) from taking more sensitive information and writing it into a less sensitive document. In this way, the BLP model guarantees that data from a high security level can never flow downwards to a lower security level. However, these two access control rules are not sufficient to guarantee absence of information leakage, as they do not prevent confidential information to be indirectly transmitted through system side effects. For example, a user with a low clearance may request to create a file of a given name. Suppose that a highly classified file of the same name already exists, then the system might reply "request denied" or, arguably worse, "request denied, a file with this name already exists". This represents a so-called covert channel that could potentially be exploited to signal high-level information to a low process. The BLP model does not prevent *indirect* information leakage due to the presence of shared resources. It is then necessary to integrate this discipline with a covert channel analysis, considering the whole flow of information.

In [5], we investigated how to express the BLP model in the Mobile Ambients framework. To this aim, we introduced the notion of *security boundary*, that allows us to identify ambients that may guarantee to properly protect their content. The intuition is the following: a boundary separates the untrusted environment from the trusted entities and data. Depending on the context, a boundary may represent different security mechanisms. For example, it may model cryptography, which protects data sent on public networks, or it may represent a protected part of a file-system where confidential data are stored in order to enforce some access control policy. Once the concept of security boundary is introduced, it becomes easy to define what absence of (direct) information leakage means: at run-time, every high-level data or process should be always encapsulated into a boundary. Thus, a direct flow is defined as a boundary crossing and may be statically detected by a suitable nesting analysis.

As a first example of direct information flow, consider the following process,

that models how a bank may communicate a credit card number:

$$bank\_db[\![\ cc\_number[\,\mathbf{out}\ bank\_db.Q\,]\ \mid P\,]\!]\,.$$

The [ ] notation denotes an ambient, while [[ ]] is used to denote security boundaries. The **out** capability, when it applies, moves the enclosing ambient out of the target ambient *bank_db*, reaching a state of the system where the credit card number is exposed to the possibly untrusted environment:

$$bank\_db[\![\,P\,]\!]\ \mid\ cc\_number[\,Q\,]\,.$$

Notice that the flow illustrated above might happen after many interactions with the bank database have been performed. As a consequence, it could be non-trivial to discover the flaw by checking all possible executions, thus the importance of an automatic method for the detection of these kinds of flows.

In this paper, we go one step further, facing the issue of detecting also *indirect* information leakage. Consider the following example:

$$bank\_db[\![\ cc\_number[\,P\,]\ \mid \mathbf{open}\ cc\_number.\ signal[\,\mathbf{out}\ bank\_db.S\,]\ \mid Q\,]\!]\,.$$

As the credit card number is not moving out of the database, in this case no direct information leakage ever arises. Instead, the presence of the credit card number in the database may be tested through an **open** capability which can be activated only if the target ambient is present as sibling and, only after that, a low-level signal is sent out of the database. The reached state is:

$$bank\_db[\![\,P \mid Q\,]\!]\ \mid\ signal[\,S\,]\,.$$

Since *signal* is low-level, this does not constitute a direct information leakage. However, we know that ambient *signal* exits the database only if the credit card number *cc_number* is present, i.e., the presence of *signal* at the environment level is caused by a confidential datum in the database. For this reason, process $\mathbf{open}\ cc\_number.\ signal[\,\mathbf{out}\ bank\_db.S\,]$ can be seen as a Trojan Horse program that has been erroneously downloaded and run inside the bank database, which indirectly leaks high-level information.

In order to face both direct and indirect information flows, we proceed as follows. First, we extend the language by including boundaries in the semantics. The advantage of this choice is that it allows to enforce a simple access control policy that guarantees the absence of direct information flow, still leaving open the much more intriguing issue of facing indirect information flow. Informally, there is no confidential data leakage (both direct and indirect) if the system behaviour is not influenced by high-level values/processes, i.e, if the high-level part of the system is not able to "interfere" with the low-level one [22]. We formalise this idea as in [1] by requiring that an interference-free process is

equal to the same process in which the names of high-level ambients have been modified.

For instance, assume that the high-level value *cc_number* of the example above, is changed into *cc_number′*: [1]

$$bank\_db[\![\ cc\_number'[\,Q\,]\ \mid \mathbf{open}\, cc\_number.\, signal[\,\mathbf{out}\, bank\_db.S\,]\ \mid P\,]\!]\,.$$

Since the **open** capability cannot be performed anymore, the *signal* ambient cannot move out the database. This difference is observable by a low-level observer, and it makes the initial process not equivalent to the perturbed one. This is due to the fact that there exists a causality between the actual high-level values and the behaviour of *signal*. This casual influence gives rise to an implicit information flow.

In order to automatically verify if a process is interference-free, we extend the control flow analysis of [5] in two directions:

(i) the presence of boundary names in the language allows us to improve the accuracy of the analysis, as constraints on boundary crossings reduce the occurrences of possible nestings and prevent direct leakage to happen;
(ii) a set of *suspect* ambients is computed, which contains all high-level ambients and is closed under the following condition: every ambient that may exercise a capability on a *suspect* ambient is *suspect* too.

Informally, this set of *suspect* ambients is an over-approximation of ambients whose behaviour could be influenced by high-level ambients. We prove that if the set of *suspect* ambients is protected inside security boundaries, then the observable behaviour of the system does not change, i.e., there is no indirect leakage.

The main contributions of our paper can be summarized as follows:

- Boundary Ambients: we extend Mobile Ambients by including in the semantics the notion of boundaries. This provides a simple translation of the BLP access control model into a core model of mobility. As mentioned above, the concept of boundary has been first introduced in [5] to model BLP rules and capture direct information leakage. However, in that work, boundaries had no semantic import and were only used to check BLP violations and not to enforce the BLP access control rules, as done in the new calculus presented here;
- non-interference: we formalize non-interference in the Boundary Ambient setting. As discussed above, this is done similarly to [1]. We will see, however,

[1] We are assuming that the Trojan Horse **open** *cc_number*. *signal*[**out** *bank_db.S*] is not part of the initial process, this is why it is not affected by the substitution.

that our setting requires some technical subtleties in order to restrict the set of observational contexts to the "well-behaving" ones, i.e., the ones that do not leak information. This non-interference notion is orthogonal to the idea of boundaries and should scale, with no substantial modifications, to different settings, e.g., to other variants of Mobile Ambients.
- control flow analysis for non-interference: we extend the control flow analysis of [5] in order to track potential causal relations among high and low level ambients. This is done through the notion of suspect ambients introduced above. The analysis of [5] is an extension of [28] in which we separate the set of nestings occurring inside or outside a boundary, achieving a more accurate over-approximation of the actual process behaviour. The extension presented here would also work on the basic nesting analysis of [28], but with strictly less precision, as we will show in Section 4.

To the best of our knowledge, there are no results in literature concerning indirect information leakage detection in the communication-free fragment Mobile Ambients [3]. The only related work we are aware of, follows a different approach aiming at defining a type system that guarantees non-interference in Boxed Ambients [13]. Moreover, the Control Flow Analysis approach allows us to infer an over-approximation of ambient nestings and suspect ambients. Thus, it leads to positive information also when process P is not recognised as interference-free, as it may dramatically reduce the size of code inspection either to find possible causes of information leakage, or to recognise it as a false positive. This is valuable when compared with the verification approach by prescriptive rules like in type-system approaches.

The rest of the paper is organised as follows. In Section 2, we introduce the Boundary Ambient calculus and we formalise multi-level security in this setting. In Section 3, we define the notions of direct and indirect information leakage, whereas in Section 4, we introduce a control flow analysis to statically verify absence of indirect information leakage in the Boundary Ambients framework. Section 5 concludes the paper with final remarks and comparisons with related works.

2 The Boundary Ambient Calculus

The Mobile Ambient calculus was introduced in [12] with the main purpose of explicitly modelling mobility. The notion of ambient captures simply and powerfully the structure and properties of wide-area networks, mobile computing, and mobile computation. Ambients are arbitrarily nested boxes which can move around through suitable capabilities. *Boundary Ambients* (*B-Ambients*, for short) extend Mobile Ambients with special ambients, called *boundaries*, that are responsible of confining confidential information, thus enforcing an

access control mechanism.

In this section, we introduce the syntax and the semantics of the B-Ambient calculus. Then, we introduce the Morris-style contextual equivalence [27] for the ambient calculus as a way of equating process behaviours. Finally, by exploiting the notion of boundary and a simple labelling of the core syntax primitives, we describe how to formalise the Bell-LaPadula model [22] in the setting of Mobile Ambients.

2.1 Syntax and Operational Semantics

The syntax of processes is the same of Mobile Ambients, except for the set of ambient names **Names**, which is partitioned into two disjoint sets, **Amb** and **Bound**: **Amb** represents the set of all ambient names and **Bound** represents the set of all boundary names. The syntax is given in Figure 1, where $n \in$ **Names**, i.e., n is either an ambient or a boundary.

Intuitively, the restriction $(\nu n)P$ introduces the new (and unique) name n and limits its scope to P; process $\mathbf{0}$ is the null process (no action) [2] ; $P \mid Q$ indicates P and Q running in parallel; replication is a technically convenient way of representing recursion and iteration as $!P$ denotes any number of copies of P in parallel. An ambient/boundary is written $n[\,P\,]$ with n the name of the ambient/boundary, and P the process running inside. In the following, when writing $n[\,P\,]$ we implicitly assume that n is an ambient, i.e., $n \in$ **Amb**, whereas the notation $n[\![\,P\,]\!]$ will be used to denote the fact that n is a boundary, i.e., $n \in$ **Bound**. The capabilities $\mathbf{in}\, n$ and $\mathbf{out}\, n$ move their enclosing ambients/boundaries in and out n, respectively; the capability $\mathbf{open}\, n$ is used to dissolve a sibling ambient/boundary n. The novelty of the calculus is the fact that moves over boundaries are controlled: $\mathbf{out}\, n$ and $\mathbf{open}\, n$, when n is a boundary, are allowed only when the executing ambient is itself a boundary. These requirements are enforced by side-conditions on the reduction rules for the *open* and the *out* capabilities, as reported in Figure 2. The only name binding operator is ν: names that are not bound by a ν operator are thus free names. We denote by $fn(P)$ the set of free names of process P, and by $bn(P)$ the set of bound names.

The operational semantics of a process P is given through a reduction relation $\rightarrow$ and a structural congruence $\equiv$ between processes. They are depicted in Figures 2 and 3, respectively. Reduction is defined by a family of inference rules. Intuitively, $P \rightarrow Q$ represents the possibility for P of reducing to Q through some computation. We will write $P \rightarrow^* Q$ to denote, as usual, the

[2] For the sake of readability, we will sometimes omit the terminating $\mathbf{0}$ at the end of every process specification, e.g., we will write $\mathbf{in}\, n \mid n[\,]$ in place of $\mathbf{in}\, n.\mathbf{0} \mid n[\,\mathbf{0}\,]$.

reflexive and transitive closure of $P \rightarrow Q$. The structural congruence, as for Mobile Ambients, rearranges the syntax of a Boundary Ambient process in order to bring potential interactors together. In addition, we identify processes up to renaming of bound names: $(\nu n)P = (\nu m)P\{n \leftarrow m\}$ if $m \notin fn(P)$. This means that these processes are understood to be identical, as opposite to structurally equivalent.

Example 2.1 Let P_1 be a process modelling a *query* sent from a *client* to a *database*:

$$\begin{aligned} P_1 = \; & \mathit{client}[\, \mathit{query}[\, \mathbf{out}\, \mathit{client}.\mathbf{in}\, \mathit{database}.Q\,]\,] \\ & \quad | \\ & \mathit{database}[\, \mathbf{open}\, \mathit{query}.R\,]\,. \end{aligned}$$

Initially, *query* is inside *client*. Then, it exits the client and enters the database by applying its capabilities $\mathbf{out}\, client$ and $\mathbf{in}\, database$, respectively. Thus, process P_1 moves to:

$$\mathit{client}[\,\mathbf{0}\,] \mid \mathit{database}[\, \mathit{query}[\,Q\,] \mid \mathbf{open}\, \mathit{query}.R\,]\,.$$

Once the database has received the query, it reads its content by consuming its $\mathbf{open}\, query$ capability. At this point, P_1 reaches the state $\mathit{client}[\,\mathbf{0}\,] \mid \mathit{database}[\,Q \mid R\,]$. The query is then processed by the interaction of Q and R. Observe that in the example there is no restriction over the moves because none of the actors in P_1 is a boundary. $\diamond$

Example 2.2 We can now exploit boundaries in order to model a simple access control mechanism. In some situations, critical operations need to be executed in a secure and protected environment. For example, let P_2 be a process modelling a *safe* inside a bank caveau *caveau1*, i.e.,

$$\begin{aligned} P_2 = \; & \mathit{caveau}1[\![\, \mathit{safe}[\![\, \mathbf{out}\, \mathit{caveau}1.\mathbf{in}\, \mathit{caveau}2 \mid Q\,]\!]\,]\!] \\ & \quad | \\ & \mathbf{open}\, \mathit{safe} \\ & \quad | \\ & \mathit{caveau}2[\![\, \mathbf{open}\, \mathit{safe}\,]\!]\,. \end{aligned}$$

Notice that, in this case, *caveau1*, *caveau2* and *safe* are boundaries, thus, according to the semantics of B-Ambients, the $\mathbf{open}\, safe$ capability can be performed only when *safe* is inside one of the caveaux, which prevents *safe* to

be opened by any ambient at the environment level. Thus, P_2 moves to

$$\begin{array}{l} P_2 = \ caveau1[\![\, \mathbf{0} \,]\!] \\ \qquad\quad | \\ \qquad \mathbf{open}\ safe \mid \ safe[\![\, \mathbf{in}\ caveau2 \mid Q \,]\!] \\ \qquad\quad | \\ \qquad caveau2[\![\, \mathbf{open}\ safe \,]\!]\,, \end{array}$$

but the *open* capability cannot be performed on the boundary *safe* at the environment level. For this reason, the only possible reduction leads the system to:

$$\begin{array}{l} P_2 = \ caveau1[\![\, \mathbf{0} \,]\!] \\ \qquad\quad | \\ \qquad \mathbf{open}\ safe \\ \qquad\quad | \\ \qquad caveau2[\![\, \mathbf{open}\ safe \mid \ safe[\![\, Q \,]\!] \,]\!] \end{array}$$

where, *safe* can finally be opened. ◇

2.2 Observational Equivalences

We now recall some definitions in [23] about contexts and observables that will be useful in the following sections. In fact, the notion of information leakage we will introduce in Section 3.2 is based on a form of Morris-style *contextual equivalence* [27] (otherwise known as may-testing equivalence) for the ambient calculus. Two processes are contextually equivalent if and only if they admit the same observations whenever they are inserted inside any arbitrary context. In the setting of the ambient calculus, contextual equivalence is defined in terms of observing the presence, at the top-level of a process, of an ambient/boundary whose name is not restricted.

Definition 2.3 (Context: $\mathcal{C}$) *A* context $\mathcal{C}$ *is a process containing zero or more holes. In the following, we write* $\mathcal{C}(P)$ *for the outcome of filling each hole in the context* $\mathcal{C}$ *with process* P*.*

Example 2.4 Let P_3 be a process of the form $P_3 = s[\,Q\,] \mid b[\![\, R \,]\!]$, and $\mathcal{C}$ a context of the form $\mathcal{C}(\_) = t[\,S\,] \mid d[\![\, \_ \,]\!]$. Then, $\mathcal{C}(P_3) = t[\,S\,] \mid d[\![\, s[\,Q\,] \mid b[\![\, R \,]\!] \,]\!]$. Notice that names which are free in P_3 may become bound in $\mathcal{C}(P_3)$. Hence, we do not identify contexts up to renaming of bound names. ◇

The following definitions formally introduce the notion of what may be observable in a B-Ambient process, i.e., ambient/boundary names not restricted at the top-level. This notion is then exploited to define to well-known process equivalences, namely contextual equivalence and barbed congruence.

Definition 2.5 (Exhibition of a Name: $P \downarrow n$) *Let P be a process, and $n \in$* **Names** *the name of either a boundary or an ambient. Then, P* exhibits name n *($P \downarrow n$) iff there are $m_1, m_2, ..., m_k$ with $m_i \neq n \ \forall i \leq k$, and two processes P' and P'' such that $P \equiv (\nu m_1, m_2, ..., m_k)(\ n[\,P'\,] \mid P'')$.*

Furthermore, a process P is said to *converge to a name $n \in$* **Names** (written $P \Downarrow n$) if P exhibits n after some parameterised reductions, i.e., if $P \rightarrow^* Q$ and $Q \downarrow n$.

The following definition is a parameterised version of the contextual equivalence described in [23] which considers only a subset $\mathbb{C}$ of all possible contexts.

Definition 2.6 (Contextual Equivalence up to $\mathbb{C}$: $P \simeq_{\mathbb{C}} P'$) *Let $\mathbb{C}$ be a set of contexts. Two processes P and P' are* contextually equivalent *up to $\mathbb{C}$, denoted ($P \simeq_{\mathbb{C}} P'$), iff for all $n \in$* **Names** *and $\mathcal{C} \in \mathbb{C}$, $\mathcal{C}(P) \Downarrow n \Leftrightarrow \mathcal{C}(P') \Downarrow n$.*

Example 2.7 To show that two processes are contextually inequivalent, it suffices to find a context that distinguishes them. For example, let $\mathcal{C}(\_) = m[\,\_\,]$ be a context in $\mathbb{C}$. If $m \neq n$, then $\mathcal{C}(\,p[\,\mathbf{out}\ m.\mathbf{0}\,]\,) \Downarrow p$, but $\mathcal{C}(\,p[\,\mathbf{out}\ n.\mathbf{0}\,]\,) \not\Downarrow p$. Thus, $p[\,\mathbf{out}\ m.\mathbf{0}\,] \not\simeq_{\mathbb{C}} p[\,\mathbf{out}\ n.\mathbf{0}\,]$. $\diamond$

In general, it is hard to prove that two processes are contextually equivalent, since one must consider their behaviour when placed in an arbitrary context. In the technical proofs we will use the following definition of barbed bisimilarity and barbed congruence, since barbed congruence is a sufficient condition for proving contextual equivalence as shown in Proposition 2.10.

Definition 2.8 (Barbed bisimilarity: $P \approx P'$) *A* barbed bisimulation *is a symmetric relation $\mathcal{S}$ such that whenever $(P, P') \in \mathcal{S}$,*

- *$P \downarrow n$ implies $P' \downarrow n$;*
- *$P \longrightarrow Q$ implies that $\exists Q'$ such that $P' \longrightarrow^* Q'$ and $(Q, Q') \in \mathcal{S}$.*

Barbed bisimilarity is the union of all barbed bisimulations. In other words, two processes P and P' are barbed bisimilar ($P \approx P'$) iff there exists a barbed bisimulation $\mathcal{S}$ such that $(P, P') \in \mathcal{S}$.

Definition 2.9 (Barbed congruence up to $\mathbb{C} : P \approx_{\mathbb{C}} P'$) *Let $\mathbb{C}$ be a set of contexts. Two processes P and P' are* barbed congruent *up to $\mathbb{C}$, denoted ($P \approx_{\mathbb{C}} P'$), iff for all $\mathcal{C} \in \mathbb{C}$, $\mathcal{C}(P) \approx \mathcal{C}(P')$.*

The following result shows that, as expected, barbed congruence is stronger than contextual equivalence, i.e., $\approx_{\mathbb{C}}$ is a sound proof-technique for contextual equivalence.

Proposition 2.10 *Let $\mathbb{C}$ be a set of contexts. $P \approx_{\mathbb{C}} P'$ implies $P \simeq_{\mathbb{C}} P'$.*

Proof. Assume $P \approx_{\mathbb{C}} P'$ and consider a context $\mathcal{C} \in \mathbb{C}$ and a name $n \in \mathbf{Names}$. We prove that $\mathcal{C}(P) \Downarrow n$ implies $\mathcal{C}(P') \Downarrow n$. $\mathcal{C}(P) \Downarrow n$ means that $\exists Q$ s.t. $\mathcal{C}(P) \rightarrow^* Q \wedge Q \downarrow n$. Since $P \approx_{\mathbb{C}} P'$ and $\mathcal{C} \in \mathbb{C}$ we also have $\mathcal{C}(P) \approx \mathcal{C}(P')$. By definition of barbed bisimilarity, $\exists Q'$ such that $\mathcal{C}(P') \rightarrow^* Q'$, with $Q \approx Q'$, thus $Q \downarrow n$ implies $Q' \downarrow n$, hence proving that $\mathcal{C}(P') \Downarrow n$.

The fact that $\mathcal{C}(P') \Downarrow n$ implies $\mathcal{C}(P) \Downarrow n$ may be symmetrically proved. We thus have that for all $\mathcal{C} \in \mathbb{C}$ and for all names $n \in \mathbf{Names}$, $\mathcal{C}(P) \Downarrow n$ iff $\mathcal{C}(P') \Downarrow n$, and so the thesis $P \simeq_{\mathbb{C}} P'$. □

2.3 Modelling Multi-level Security

Let us now formalise the Bell-LaPadula model in the setting of Mobile and Boundary Ambients. Figure 4 summarises all the flows allowed by the BLP model, including covert channels. Security levels are arranged into a lattice $\langle L, \leq \rangle$, where $\ell_1 \leq \ell_2$ means that ℓ_1 has a security level lower than ℓ_2, i.e., ℓ_2 dominates ℓ_1. For the sake of simplicity, we use a simple lattice $\langle L, \leq \rangle$ with $L = \{\mathsf{low}, \mathsf{high}\}$ and $\mathsf{low} \leq \mathsf{high}$. In order to express the Bell-LaPadula model in the context of Mobile Ambients, the following issues must be taken into account:

(1) both subjects and objects must be classified into different security levels;
(2) the *No-Read up* and *No-Write down* rules must be interpreted and formalised in the Mobile Ambients framework, specifying how `read`/`write` actions are implemented.

As it is customary in static analysis, labels $\ell^a \in \mathbf{Lab}^a$ on boundaries and ambients, and labels $\ell^t \in \mathbf{Lab}^t$ on transitions are introduced, both to determine the program points of interest during the analysis computation, and to assign different security levels to ambients. The syntax of processes is then enriched with labels as follows, where only the affected primitives are reported. Semantics is changed accordingly.

$$
\begin{array}{lcll}
P, Q & ::= & \cdots & \\
& | & n^{\ell^a}[\,P\,] & \text{ambient or boundary} \\
& | & \mathbf{in}^{\ell^t}\, n.P & \text{capability to enter } n \\
& | & \mathbf{out}^{\ell^t}\, n.P & \text{capability to exit } n \\
& | & \mathbf{open}^{\ell^t}\, n.P & \text{capability to open } n
\end{array}
$$

The set of ambient labels $\mathbf{Lab}^a$ is partitioned into three disjoint sets:

- $\mathbf{Lab}^a_H$, labels of ambients classified high,
- $\mathbf{Lab}^a_L$, labels of ambients classified low,
- $\mathbf{Lab}^a_B$, labels of *boundaries*, which are neither low or high since they only aim at confining confidential information.

In all the examples, we will use the following notation for labels: $b \in \mathbf{Lab}^a_B, h \in \mathbf{Lab}^a_H$, $m \in \mathbf{Lab}^a_L$ and $c \in \mathbf{Lab}^t$, and the special label $env \in \mathbf{Lab}^a_L$ to represent the external environment. In the rest of the paper, we will also assume that the ambient and capability labels occurring in a process P are all distinct and consistent, i.e., an ambient n is never labelled both as high and as low. Performing the Control Flow Analysis with all distinct labels produces a more precise result that can be later approximated by equating some labels.

Since we consider the communication-free fragment of MA, we regard the ambient itself both as a subject and an object, i.e., both as an active member performing a `read`/`write` operation, and as a passive resource. This means that both data, processes and locations are abstractly represented as ambients. Moreover, there are not explicit `read` and `write` actions, so they will be expressed in terms of capability actions.

As formalised by the barbed congruence notion presented in the previous section, an ambient is observable (i.e., accessible) only at the environment level. Thus, in order to protect high-level ambients, we always enclose them inside boundaries. In fact, the semantics of boundaries enforces two access rules that intuitively correspond to the BLP ones:

- a boundary can only be opened by another boundary: this forbids boundaries to be dissolved by external low level ambients;
- only boundaries can exit other boundaries: this forbids a high-level ambient to exit its protective boundary.

Notice that the first rule controls access from low to high while the second one regulates access from high to low. Intuitively, we can read these two rules as the BLP *No Read up* and *No Write down*. Entering a boundary is always allowed modelling unrestricted information flow from low to high.

Example 2.11 As an example, consider the following labelled process:

$$P_4 = \mathit{container}^{b1}[\![\ \mathit{hdata}^{h}[\,\mathbf{out}^{c1}\,\mathit{container}\,]\ \mid \mathit{send}^{b2}[\![\,\mathbf{out}^{c2}\,\mathit{container}\,]\!]\ ]\!]\,.$$

This process is an example of how boundaries may prevent *direct* information flow. Ambient *container* is a boundary protecting high level data *hdata* (note that data are abstractly represented as ambients): ambient *hdata* cannot perform the *out* capability from boundary *container*, as it is prevented by the semantics of B-Ambients. For this reason, *hdata* is never exposed to any ambient at environment level.

On the other hand, *send*, which is a boundary, may go out of *container* without causing any direct information leakage. In particular, P_4 may only evolve to the state:

$$\begin{array}{l}\mathit{container}^{b1}[\![\ \mathit{hdata}^{h}[\,\mathbf{out}^{c1}\,\mathit{container}\,]\ ]\!] \\ \quad\mid \\ \mathit{send}^{b2}[\![\,\mathbf{0}\,]\!]\,.\end{array}$$

◇

3 Information Flow

The access control modelled in the ambient calculus is rather naive: a process may move into or out of a particular ambient only if it owns the appropriate capability. The notion of security boundary can be seen as a stricter access control mechanism, where some capability can be used only in a "secure" controlled way. *Direct* information flow is avoided "by construction" in the B-Ambient calculus. The intuition is the following: absence of direct information flow is guaranteed if, at run-time, every high-level datum or process is encapsulated into at least one boundary ambient. In B-Ambients only boundaries may exit from, or open, boundaries, thus high-level ambients always remain protected during the process execution if they are encapsulated inside a boundary since the beginning. The problem of detecting *indirect* information flow is more subtle: implicit information flow results from transmitting information via system side effects. In this case, the casual chain of events needs to be detected, since a low-level action may depend on the presence of a high-level ambient.

This section is devoted to the formalisation of these concepts. The first subsection discusses direct information leakage, while the second one defines indirect information leakage in the B-Ambients setting.

The flow of high-level ambients outside security boundaries is the downward flow that the Bell-LaPadula model intends to avoid. It may be formalised as follows. In the definition, we use both the function $Nest_\ell$ reported in Figure 5 and the predicate $\texttt{Unprotected}_{\ell'}(\ell, R)$ described below.

The function $Nest_\ell$ collects all the nestings among ambients/boundaries and capabilities of a given process P with respect to an enclosing ambient/boundary labelled with ℓ: if P contains at top-level either an ambient/boundary labelled ℓ^a or a capability labelled ℓ^t, then the pair (ℓ, ℓ^a) or (ℓ, ℓ^t), respectively, is added to the result of the function.

Given an ambient/boundary labelled ℓ and a set $R \subseteq (\mathbf{Lab}^a \times (\mathbf{Lab}^a \cup \mathbf{Lab}^t))$ representing nestings among ambients and capabilities, the predicate $\texttt{Unprotected}_{\ell'}(\ell, R)$ asserts that there is at least one path in R from ℓ' down to ℓ, in which none of the labels is of a boundary. In other words, there is no boundary protecting ℓ with respect to the enclosing ambient/boundary labelled ℓ'. In the following, we will write $\texttt{Unprotected}(\ell, R)$ whenever $\ell' = env$.

Definition 3.1 (Unprotected) *Given a labelled process P and a set $R \subseteq (\mathbf{Lab}^a \times (\mathbf{Lab}^a \cup \mathbf{Lab}^t))$, $\texttt{Unprotected}_{\ell'}(\ell, R)$ = true iff $\exists\ \ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B(P)$ s.t $(\ell', \ell_1), (\ell_1, \ell_2), ..., (\ell_{n-1}, \ell_n), (\ell_n, \ell) \in R$.*

On the converse, the predicate $\texttt{Protected}_{\ell'}(\ell, R) = \neg\texttt{Unprotected}_{\ell'}(\ell, R)$ asserts that an ambient labelled ℓ is protected only if it is contained inside at least one boundary within ℓ'. The above notion of `Protected` and `Unprotected` is extended to paths as follows: a *path* $\mathcal{P} = \ell_1, ..., \ell_n$ is unprotected (protected) if $\forall i \in [1, n]$, $\ell_i \notin \mathbf{Lab}^a_B$ ($\exists i \in [1, n]$ s.t. $\ell_i \in \mathbf{Lab}^a_B$). To simplify the notation, we often write (`Un`)`Protected` (ℓ, P) in place of (`Un`)`Protected` $(\ell, Nest_{env}(P))$.

Example 3.2 Consider again process P_4 of Example 2.11. The function $Nest_{env}(P_4)$ computes all the nestings among ambients/boundaries and capabilities of process P_4 with respect to the external environment, that is $Nest_{env}(P_4) = \{(env, b1), (b1, h), (h, c1), (b1, b2), (b2, c2)\}$. It is easy to verify that $\texttt{Protected}(h, P_4)$ is true, as h is protected inside boundary $b1$. $\diamond$

Intuitively, a process P directly leaks information if, in at least one of its possible executions, one of its high-level ambients results to be unprotected with respect to the external environment env.

Definition 3.3 (Direct Information Leakage) *Given a labelled process P, P directly leaks secret $h \in \mathbf{Lab}^a_H$ iff $\exists Q, P \rightarrow^* Q$ such that* `Unprotected` (h, Q).

We can prove that any high level ambient which is initially nested inside (at

least) one boundary, is never leaked at run-time, i.e., B-Ambient access control rules forbid direct information leakage.

Proposition 3.4 (Absence of Direct Information Leakage) *Given a labelled B-Ambient process P and a high level label $h \in \mathbf{Lab}^a_H$, if* $\texttt{Protected}\ (h, P)$ *then P does not directly leak secret h.*

Proof. First, it can be easily proved by induction on the depth of derivations of $P \equiv Q$ that congruence between processes preserves their nestings, i.e., $Nest_\ell(P) = Nest_\ell(Q)$. Then, it is sufficient to prove by induction on the derivation of $P \rightarrow Q$ that if $\texttt{Protected}_\ell(h, Nest_\ell(P))$ and $P \rightarrow Q$, then $\texttt{Protected}_\ell(h, Nest_\ell(Q))$ with respect to an enclosing ambient/boundary labelled ℓ.

Base Step: By case analysis on the axioms of Figure 2.

(*InRed*) Let $P = n^{\ell^a}[\,\mathbf{in}^{\ell^t}\, m.P' \mid Q'\,] \mid m^{\ell^{a'}}[\,R'\,]$ and $Q = m^{\ell^{a'}}[\, n^{\ell^a}[\,P' \mid Q'\,] \mid R'\,]$. In this case, we have that:
- $Nest_{env}(P) = \{(env, \ell^a), (env, \ell^{a'}), (\ell^a, \ell^t)\} \cup Nest_{\ell^a}(P') \cup Nest_{\ell^a}(Q') \cup Nest_{\ell^{a'}}(R')$;
- $Nest_{env}(Q) = \{(env, \ell^{a'}), (\ell^{a'}, \ell^a)\} \cup Nest_{\ell^a}(P') \cup Nest_{\ell^a}(Q') \cup Nest_{\ell^{a'}}(R')$.

In other words, after the reduction step, a new pair $(\ell^{a'}, \ell^a)$ is added in $Nest_{env}(Q)$, modelling the nesting resulting after the move, while (env, ℓ^a) and (ℓ^a, ℓ^t) are absent since the capability has been consumed yielding ambient n inside m. All the other pairs are the same as in $Nest_{env}(P)$. Assume $\texttt{Protected}\ (h, Nest_{env}(P))$ with $h \in \mathbf{Lab}^a_H$. This means that there are no paths of the form $(env, \ell_1), (\ell_1, \ell_2), ..., (\ell_{n-1}, \ell_n), (\ell_n, h) \in Nest_{env}(P)$ with $\ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B$. Suppose that the new pair generates an unprotected path, i.e., $(env, \ell^{a'}), (\ell^{a'}, \ell^a), (\ell^a, \ell_1), ..., (\ell_n, h) \in Nest_{env}(Q)$ with $\ell^{a'}, \ell^a, \ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B$. In this case, also path $(env, \ell^a), (\ell^a, \ell_1), ..., (\ell_n, h) \in Nest_{env}(P)$ would be unprotected. This leads to a contradiction since we assumed that $Nest_{env}(P)$ is protected. Thus, $\texttt{Protected}(h, Nest_{env}(Q))$ holds.

(*OutRed*) Let $P = m^{\ell^{a'}}[\, n^{\ell^a}[\,\mathbf{out}^{\ell^t}\ m.P' \mid Q'\,] \mid R'\,]$ and $Q = n^{\ell^a}[\,P' \mid Q'\,] \mid m^{\ell^{a'}}[\,R'\,]$, with $m \in \mathbf{Bound} \Rightarrow n \in \mathbf{Bound}$. In this case, we have that:
- $Nest_{env}(P) = \{(env, \ell^{a'}), (\ell^{a'}, \ell^a), (\ell^a, \ell^t)\} \cup Nest_{\ell^a}(P') \cup Nest_{\ell^a}(Q') \cup Nest_{\ell^{a'}}(R')$;
- $Nest_{env}(Q) = \{(env, \ell^a), (env, \ell^{a'})\} \cup Nest_{\ell^a}(P') \cup Nest_{\ell^a}(Q') \cup Nest_{\ell^{a'}}(R')$.

In other words, after the reduction step, a new pair (env, ℓ^a) is added in $Nest_{env}(Q)$, modelling the nesting resulting after the move, while $(\ell^{a'}, \ell^a)$ and (ℓ^a, ℓ^t) are absent since the capability has been consumed yielding ambient n out of m. All the other pairs are the same as in $Nest_{env}(P)$. Let us assume $\texttt{Protected}(h, Nest_{env}(P))$ with $h \in \mathbf{Lab}^a_H$, i.e., there are no paths of the form $(env, \ell_1), (\ell_1, \ell_2), ..., (\ell_{n-1}, \ell_n), (\ell_n, h) \in Nest_{env}(P)$ with $\ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B$. If n is a boundary, then $\ell^a \in \mathbf{Lab}^a_B$, and the only new pair added to $Nest_{env}(Q)$, i.e., (env, ℓ^a), cannot create any unprotected

path from env to any h. If n is not a boundary, then also m is not a boundary, and $\ell^a, \ell^{a'} \notin \mathbf{Lab}^a_B$. Let us suppose there is a new unprotected path $(env, \ell^a), (\ell^a, \ell_1), ..., (\ell_{n-1}, \ell_n), (\ell_n, h) \in Nest_{env}(Q)$, with $\ell^a, \ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B$. In this case, also path $(env, \ell^{a'}), (\ell^{a'}, \ell^a), (\ell^a, \ell_1), ..., (\ell_{n-1}, \ell_n), (\ell_n, h) \in Nest_{env}(P)$ would be unprotected, which leads to a contradiction since we assumed that $Nest_{env}(P)$ was protected. Thus, we can conclude $\texttt{Protected}(h, Nest_{env}(Q))$ holds.

(*OpenRed1*) Let, $P = n^{\ell^a}[\,\mathbf{open}^{\ell^t}\ m.P' \mid m^{\ell^{a'}}[\,Q'\,] \mid R'\,]$ and process $Q = n^{\ell^a}[\,P' \mid Q' \mid R'\,]$ with $m \in \mathbf{Bound} \Rightarrow n \in \mathbf{Bound}$. In this case, we have that:

- $Nest_{env}(P) = \{(env, \ell^a), (\ell^a, \ell^{a'}), (\ell^a, \ell^t)\} \cup Nest_{\ell^a}(P') \cup Nest_{\ell^{a'}}(Q') \cup Nest_{\ell^a}(R')$;
- $Nest_{env}(Q) = \{(env, \ell^a)\} \cup Nest_{\ell^a}(P') \cup Nest_{\ell^a}(Q') \cup Nest_{\ell^a}(R')$.

In other words, after the reduction step, the pairs $(\ell^a, \ell^{a'})$ and (ℓ^a, ℓ^t) are absent since the capability has been consumed dissolving the ambient labelled $\ell^{a'}$. As a consequence, process Q', which was contained inside the opened ambient labelled $\ell^{a'}$, has been inherited by the dissolving ambient labelled ℓ^a, i.e., in $Nest_{env}(Q)$ we have $Nest_{\ell^a}(Q')$ instead of $Nest_{\ell^{a'}}(Q')$. All the other pairs are the same as in $Nest_{env}(P)$. Assume $\texttt{Protected}\ (h, Nest_{env}(P))$ holds with $h \in \mathbf{Lab}^a_H$, i.e., there are no paths of the form $(env, \ell_1), (\ell_1, \ell_2), ..., (\ell_{n-1}, \ell_n), (\ell_n, h) \in Nest_{env}(P)$ with $\ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B$. If n is a boundary, then $\ell^a \in \mathbf{Lab}^a_B$, thus the fact that in $Nest_{env}(Q)$ there is $Nest_{\ell^a}(Q')$ instead of $Nest_{\ell^{a'}}(Q')$ cannot create any unprotected path from env to any h. If n is not a boundary, then also m is not a boundary, i.e., $\ell^a, \ell^{a'} \notin \mathbf{Lab}^a_B$. Thanks to this condition, the level of protection of the paths in Q' is unchanged when the process is inside either ℓ^a or $\ell^{a'}$. Thus, $\texttt{Protected}(h, Nest_{env}(Q))$ holds.

(*OpenRed2*) Let, $P = \mathbf{open}^{\ell^t}\ m.P' \mid m^{\ell^{a'}}[\,Q'\,]$ and $Q = P' \mid Q'$, with $m \in \mathbf{Amb}$. In this case:

- $Nest_{env}(P) = \{(env, \ell^t), (env, \ell^{a'})\} \cup Nest_{env}(P') \cup Nest_{\ell^{a'}}(Q')$;
- $Nest_{env}(Q) = Nest_{env}(P') \cup Nest_{env}(Q')$.

In other words, after the reduction step, the pairs (env, ℓ^t) and $(env, \ell^{a'})$ are absent since the capability has been consumed dissolving the ambient labelled $\ell^{a'}$. For this reason, process Q', which was contained inside the opened ambient labelled $\ell^{a'}$, is at top-level, i.e., in $Nest_{env}(Q)$ we have $Nest_{env}(Q')$ instead of $Nest_{\ell^{a'}}(Q')$. All the other pairs are the same as in $Nest_{env}(P)$. Assume $\texttt{Protected}(h, Nest_{env}(P))$ with $h \in \mathbf{Lab}^a_H$, i.e., there are no paths of the form $(env, \ell_1), (\ell_1, \ell_2), ..., (\ell_{n-1}, \ell_n), (\ell_n, h) \in Nest_{env}(P)$ with $\ell_1, ..., \ell_n \notin \mathbf{Lab}^a_B$. Since m can be only an ambient, with $\ell^{a'} \notin \mathbf{Lab}^a_B$, the level of protection of the paths in Q' is unchanged when the process is inside either env or $\ell^{a'}$. Thus, $\texttt{Protected}(h, Nest_{env}(Q))$ holds.

Inductive Step: By case analysis of the last rule applied among the ones in Figure 2.

(*ResRed*) Let $P = (\nu n)P'$, $Q = (\nu n)Q'$, and $P' \rightarrow Q'$. By induction hypothesis, we have that $\texttt{Protected}_\ell(h, Nest_\ell(P'))$ implies $\texttt{Protected}_\ell(h, Nest_\ell(Q'))$. Then, since $Nest_{env}((\nu n)P') = Nest_{env}(P')$ and $Nest_{env}((\nu n)Q') = Nest_{env}(Q')$, the implication holds on P and Q as well.

(*AmbRed*) Let $P = n^{\ell^a}[\,P'\,]$, $Q = n^{\ell^a}[\,Q'\,]$, and $P' \rightarrow Q'$. Assume that $\texttt{Protected}(h, Nest_{env}(P))$ holds with $h \in \mathbf{Lab}^a_H$ and $Nest_{env}(P) = \{(env, \ell^a)\} \cup Nest_{\ell^a}(P')$. It follows that ambient n is not a high-level ambient (otherwise there would be an unprotected path from env to ℓ^a), i.e. $\ell^a \neq h$, and that $\texttt{Protected}_{\ell^a}(h, Nest_{\ell^a}(P'))$. By induction hypothesis, we have that $\texttt{Protected}_{\ell^a}(h, Nest_{\ell^a}(Q'))$. Then, since $Nest_{env}(Q) = Nest_{\ell^a}(Q') \cup \{(env, \ell^a)\}$ and $\ell^a \neq h$, $\texttt{Protected}\ (h, Nest_{env}(Q))$ holds as well.

(*CompRed*) Let $P = P' \mid R$, $Q = Q' \mid R$, and $P' \rightarrow Q'$. Assume that $\texttt{Protected}(h, Nest_{env}(P))$ with $h \in \mathbf{Lab}^a_H$ and $Nest_{env}(P) = Nest_{env}(P') \cup Nest_{env}(R)$. Then, $\texttt{Protected}(h, Nest_{env}(P'))$ and $\texttt{Protected}(h, Nest_{env}(R))$. By induction hypothesis, it holds that $\texttt{Protected}(h, Nest_{env}(Q'))$. Then, since $Nest_{env}(Q) = Nest_{env}(Q') \cup Nest_{env}(R)$, $\texttt{Protected}(h, Nest_{env}(Q))$ holds as well.

($\equiv$ *Red*) Let $P \equiv P'$ and $Q \equiv Q'$ and $P' \rightarrow Q'$. By the fact that congruence between processes preserves their nesting (i.e., $Nest_\ell(P) = Nest_\ell(P')$ if $P \equiv P'$), we have that $Nest_{env}(P) = Nest_{env}(P')$. Assume $\texttt{Protected}(h, Nest_{env}(P))$ holds, with $h \in \mathbf{Lab}^a_H$, then $\texttt{Protected}(h, Nest_{env}(P'))$ holds as well. By induction hypothesis, $\texttt{Protected}(h, Nest_{env}(Q'))$. Again, since congruence between processes preserves their nesting, $\texttt{Protected}(h, Nest_{env}(Q))$ holds as well. □

Example 3.5 Consider a simple cryptographic protocol, with two principals, *alice* and *bob*, willing to exchange some confidential information that need to be protected during the communication process. This can be modelled by defining two boundaries, one for each principal, and one high level ambient representing the confidential information. We need a mechanism for securely moving confidential data from one boundary to the other. This may be achieved through the introduction of a third boundary, *encrypt*, containing the critical data, which moves out from the first protected area and into the second one (i.e., behaving as a sort of encryption mechanism of the confidential data sent from one actor to the other). The protocol may be formalised as follows:

$$\begin{aligned} P_5 = \; & alice^{b1}[\![\; encrypt^{b2}[\![\, \mathbf{out}^{c1}\, alice.\mathbf{in}^{c2}\, bob \,]\!] \;\mid\; hdata^h[\,\mathbf{in}^{c3}\, encrypt\,]\;]\!] \\ & \mid \\ & bob^{b3}[\![\, \mathbf{open}^{c4}\, encrypt \mid Q \,]\!]\,. \end{aligned}$$

The confidential datum *hdata* has the capability to enter *encrypt* ambient, which then migrates out of its parent ambient *alice* and inside the sibling ambient *bob*. Process P_5 may then evolve to the following state (see steps (a)

to (d) of Figure 6):

$$\begin{array}{l} alice^{b1}[\![\ ]\!] \\ \quad | \\ bob^{b3}[\![\, \mathbf{open}^{c4}\, encrypt \mid Q \mid\ encrypt^{b2}[\![\ hdata^{h}[\ ]\ ]\!]\ ]\!]\,, \end{array}$$

and, finally, through step (e) of Figure 6, it reduces to:

$$alice^{b1}[\![\ ]\!] \mid bob^{b3}[\![\, Q \mid\ hdata^{h}[\ ]\ ]\!]\,.$$

Observe that *encrypt* is labelled as a boundary. Thus, the high-level datum *hdata* is protected by at least one boundary ambient during the whole execution of the process, i.e., `Protected` (h, Q) is true for all Q such that $P_5 \rightarrow^* Q$. If it was not a boundary, it could not exit from *alice* because of the *No Write down* access control rule. Furthermore, notice that *alice* could send an empty message to *bob* in case the *encrypt* ambient moves out of *alice* before *hdata* enters it: such a situation can be avoided by making the specification of the protocol more deterministic. ◇

3.2 Indirect Information Flow

In this section, we turn to implicit information flows. To this aim, we follow the standard approach based on non-interference [22]. Informally, there is no information flow from high to low if and only if the system behaviour is not influenced by high-level values/processes, i.e, if and only if the high-level part of the system is not able to influence the low-level one (see, e.g., [18,19,21] for more detail on non-interference-based properties).

Example 3.6 Let P_6 be the following process:

$$\begin{array}{rl} P_6 = & container^{b1}[\![\ send^{b2}[\![\, \mathbf{in}^{c1}\, hdata.\mathbf{out}^{c2}\, hdata.\mathbf{out}^{c3}\, container\,]\!]\ \mid \\ & \qquad \mathbf{open}^{c4}\, download\,]\!]\,. \end{array}$$

In this process, *send* may exit from *container* because of the presence of *hdata* (e.g., in a downloaded application which is opened once it enters *container*). There is no direct flow, i.e., no high-level ambient exits *container*, but a low-level user may deduce information about the presence of a high-level ambient in an implicit way, by observing the overall system behaviour. ◇

A formal definition of absence of information leakage based on non-interference can be obtained by adopting the approach introduced in [1]. The key idea is that any perturbation, i.e. a substitution of high-level ambients' names, should not affect the observable behaviour of the system. If this happens, we should be guaranteed that no interference is possible from level high to low.

The perturbation is obtained by means of a substitution σ_N which is a function that maps names in a set N to different names, and leaves all the other names unchanged.

Definition 3.7 (Substitution Function σ_N) *Let $N \in$ **Names** be a set of names. A substitution function σ_N over N is a function σ_N : **Names** $\rightarrow$ **Names**, such that $\sigma_N(s) = s$, whenever $s \notin N$. We denote with $P\sigma_N$ the process P in which σ_N is applied to all the name occurrences.*

The effect of applying a substitution σ_N to a process P is essentially to replace each free occurrence of each name in P. However, the replacement must be done in such a way that unintended capture of names by binders is avoided, i.e., we assume that the bound names of processes are chosen to be different from their free names and from the names of the substitutions.

If the set N represents a set of names that should be protected, i.e., high-level ambients, we may compare the behaviour of the system before and after the perturbation using the contextual equivalence introduced in Section 2.2, as follows.

Definition 3.8 (Absence of Indirect Information Leakage) *Let P be a process, $N \subseteq$ **Names** be a set of names and $\mathbb{C}$ be a set of contexts. P does not leak secrets N to $\mathbb{C}$ if and only if, for all substitution functions σ_N, we have $P \simeq_{\mathbb{C}} P\sigma_N$.*

Intuitively, N represents a set of names that should be protected, i.e., high-level ambients. Given a set of contexts $\mathbb{C}$, we say that P does not leak high-level information to $\mathbb{C}$ if and only if any perturbation of such information is not visible whenever P is executed in every context $\mathcal{C} \in \mathbb{C}$. If this is the case, even after the perturbation, the names that can be observed at the top-level of the process during the whole execution are the same, i.e., the observational behaviour of P remains unchanged.

Example 3.9 Consider again process P_6 of Example 3.6. We show that it leaks name *hdata* by finding a context that distinguishes P_6 from $P_6\sigma_{\{hdata\}}$. Consider context $\mathcal{C}(\_) = \_ \mid download^{b3}[\![\, \mathbf{in}^{c5}\, container \mid hdata^{h}[\, \mathbf{0}\,]\,]\!]$, and substitution $\sigma_{\{hdata\}}(hdata) = hdata'$. Then, process $\mathcal{C}(P_6)$

is as follows:

$$\begin{aligned}\mathcal{C}(P_6) = \; & container^{b1}[\![\; send^{b2}[\![\, \mathbf{in}^{c1}\, hdata.\mathbf{out}^{c2}\, hdata.\mathbf{out}^{c3}\, container \,]\!] \;\mid \\ & \qquad\qquad \mathbf{open}^{c4}\, download \,]\!] \\ & \mid \\ & download^{b3}[\![\, \mathbf{in}^{c5}\, container \mid\; hdata^{h}[\,\mathbf{0}\,] \,]\!]\,.\end{aligned}$$

It is easy to verify that $\mathcal{C}(P_6) \Downarrow send$, since after *download* enters *container* and it is opened, *send* can perform the in and out capabilities over *hdata*, and then exit at top-level. On the contrary, in process $\mathcal{C}(P_6\sigma_{\{hdata\}})$, ambient *send* is blocked inside *container*:

$$\begin{aligned}\mathcal{C}(P_6\sigma_{\{hdata\}}) = \; & container^{b1}[\![\; send^{b2}[\![\, \mathbf{in}^{c1}\, hdata'.\mathbf{out}^{c2}\, hdata'.\mathbf{out}^{c3}\, container \,]\!] \;\mid \\ & \qquad\qquad \mathbf{open}^{c4}\, download \,]\!] \\ & \mid \\ & download^{b3}[\![\, \mathbf{in}^{c5}\, container \mid\; hdata^{h}[\,\mathbf{0}\,] \,]\!]\,.\end{aligned}$$

In fact, after the renaming of *hdata* into *hdata′*, ambient *send* is stuck, trying to perform the in and out capabilities over *hdata′* before exiting *container*. Thus, *send* never reaches the top-level, i.e., $\mathcal{C}(P_6\sigma_{\{hdata\}}) \not\Downarrow send$. Therefore, we can conclude that $P_6 \not\approx_{\{\mathcal{C}\}} P_6\sigma_{\{hdata\}}$, i.e., that P_6 indirectly leaks *hdata* to $\mathcal{C}(\_)$. ◇

4 Control Flow Analysis for Information Leakage Detection

So far, we have described how direct flows are correctly prevented by suitable access control rules based on boundaries, while indirect flows are still possible. In Section 4.1, we define a control flow analysis to statically verify that a system P does not indirectly leak information. This new analysis is an extension of the one presented in [5] which, in turns, extends the one of [28]. They all aim at computing a safe approximation of the dynamic behaviour of Mobile Ambients programs. In particular, in [28], the control structure computed by the analysis is expressed by the hierarchical structure of ambients, given by the father-son relationship between the nodes of the tree structure, i.e., the analysis returns a set which contains all the ambient nestings that are possible during run-time. In [5], we have considered nesting inside and outside security boundaries separately, distinguishing between run-time nestings which are either *protected*, or *unprotected*, as formalized in Definition 3.1. This leads to a more precise (i.e., less false negatives) and a more efficient (i.e., less spuri-

ous pairs recorded) analysis with respect to [28]. In order to detect indirect information flow, we further extend the analysis in two directions:

- we add a new component collecting an over-approximation of *suspect* ambients, i.e., ambients whose behaviour may be influenced by the presence of high-level information during the run-time execution, causing information to be indirectly leaked;
- we exploit the special semantics of boundaries, forbidding direct leakages, to achieve an even more precise and efficient analysis: given that some moves are forbidden, the possible nestings in Boundary Ambients are usually less than in the standard Mobile Ambients.

The correctness of the analysis is reported in Section 4.2. In Section 4.3, we show how the security result is obtained by post-processing the result of the analysis: if the set of suspect ambients is protected during the whole run-time execution, we are guaranteed that the system is interference-free.

4.1 Specification of the Analysis for Boundary Ambients

The control flow analysis is expressed in terms of a tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$, where:

- The first component $\hat{S}$ (suspect ambients) is an element of $\wp(\lfloor \mathbf{Names} \rfloor)$, where $\lfloor \mathbf{Names} \rfloor$ denotes the set of stable names which are defined below. If a process contains an ambient n which is a high-level ambient, or whose execution is influenced by high-level ambients (e.g., which performs capabilities over suspect ambients), then n should be in $\hat{S}$.
- The second component $\hat{I}_B$ is an element of $\wp(\mathbf{Lab}^a \times (\mathbf{Lab}^a \cup \mathbf{Lab}^t))$. If a process contains either a capability or an ambient labelled ℓ inside an ambient labelled ℓ^a which is a boundary or an ambient nested inside a boundary (referred as *protected ambient* from Definition 3.1), then (ℓ^a, ℓ) is expected to belong to $\hat{I}_B$. Thus, $\hat{I}_B$ is the set of protected run-time nestings, with B standing for Boundary.
- The third component $\hat{I}_E$ is still an element of $\wp(\mathbf{Lab}^a \times (\mathbf{Lab}^a \cup \mathbf{Lab}^t))$. If a process contains either a capability or an ambient labelled ℓ inside an ambient labelled ℓ^a which is not protected, then (ℓ^a, ℓ) is expected to belong to $\hat{I}_E$. Thus, $\hat{I}_E$ is the set of unprotected run-time nestings, with E standing for External environment.
- The forth component $\hat{H} \in \wp(\mathbf{Lab}^a \times \lfloor \mathbf{Names} \rfloor)$ keeps track of the correspondence between names and labels. If a process contains an ambient or a boundary labelled ℓ^a with name n, then (ℓ^a, n) is expected to belong to $\hat{H}$.

Notice that the usage of stable names aims at keeping the analysis result finite, as, by restriction and replication, a process may generate an infinite number of different names at run-time. By following [28], we assume that α-conversion

preserves stable names, i.e., whenever n is α-converted to m we require that $\lfloor n \rfloor = \lfloor m \rfloor$. Intuitively, $\lfloor n \rfloor$ is a representative for a class of α-convertible names. For the sake of readability, we always omit the $\lfloor \cdot \rfloor$ notation in the analysis specification.

The set of solutions $\langle \{(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})\}, \sqsubseteq \rangle$ is a complete lattice, with $\sqsubseteq$ defined as $(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \sqsubseteq (\hat{S}'', \hat{I}''_B, \hat{I}''_E, \hat{H}'')$ iff $\hat{S}' \subseteq \hat{S}'' \wedge \hat{I}'_B \subseteq \hat{I}''_B \wedge \hat{I}'_E \subseteq \hat{I}''_E \wedge \hat{H}' \subseteq \hat{H}''$. The least upper bound operator, $\sqcup$, used to combine the information from the two lattices, is defined by component-wise union. By exploiting the ordering operator, we are able to compare the precision of different analyses: a "smaller" analysis is more precise than a "larger" one.

According to the control flow framework in [29], the analysis is defined by a representation function and an analysis specification. They are depicted, respectively, in Figure 7 and Figure 8.

The representation function aims at mapping concrete values to their best abstract representation. It is given in terms of a function $\beta^{\mathrm{B}}_{\ell,Proct}(P)$ which recursively builds sets $\hat{S}$, $\hat{I}_B$, $\hat{I}_E$, and $\hat{H}$ corresponding to process P, with respect to an enclosing ambient labelled with ℓ. Moreover, *Proct* is a Boolean flag which is used to indicate if the considered process is nested inside at least one boundary. In case $Proct = true$, all the forthcoming nestings will be recorded as protected in $\hat{I}_B$. The representation function $\beta^{\mathrm{B}}(P)$ of a process P is defined as $\beta^{\mathrm{B}}_{env,False}(P)$. Intuitively, function $\beta^{\mathrm{B}}(P)$ collects in $\hat{I}_B$ (in $\hat{I}_E$) all the nestings of ambients and capabilities initially (not) contained inside at least one boundary, whereas in $\hat{H}$ it records all the mappings between labels and ambients/boundaries. Finally, $\hat{S}$ collects the name of high-level ambients, by exploiting the *amb*-rule.

Example 4.1 Consider again process P_4 introduced in Section 2.3:

$$P_4 = \mathit{container}^{b1}[\![\ \mathit{hdata}^{h}[\,\mathbf{out}^{c1}\,\mathit{container}\,] \mid \mathit{send}^{b2}[\![\,\mathbf{out}^{c2}\,\mathit{container}\,]\!]\ ]\!]\,.$$

The representation function of P_4 is $\beta^{\mathrm{B}}(P_4) = (\{\mathit{hdata}\}, \{(b1, h), (b1, b2), (h, c1), (b2, c2)\}, \{(\mathit{env}, b1)\}, \{(b1, \mathit{container}), (h, \mathit{hdata}), (b2, \mathit{send})\})$. The first component records that *hdata*, a high-level ambient, is a suspect ambient, while the second and the third components capture all ambient nestings. The correspondence between ambients and labels in P_4 are kept by the last component, i.e., $\{(b1, \mathit{container}), (h, \mathit{hdata}), (b2, \mathit{send})\}$. ◇

The specification states a closure condition of a tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ with respect to all possible moves executable in a process P. The specification rules of Figure 8 depict how the process transforms one abstract representation to

another one, mimicking the process execution. They mostly amount to recursive checks of sub-processes except for the three capabilities *open*, *in*, and *out*. After briefly discussing the rules, in the next section we will prove the correctness of the analysis by showing that every reduction of the semantics is properly mimicked in the analysis. Within the specification of the analysis, the predicate $path_E(\ell^a, \ell)$ is used to simplify the notation. Intuitively, it represents the existence of an unprotected path of nestings from ambient labelled ℓ^a to ambient labelled ℓ, in which none of the ambients is a boundary. The rules for capabilities are divided in two parts: (1) the construction of sets $\hat{I}_B$ and $\hat{I}_E$, which is done by refining the analyses of [5,28] in order to correctly handle boundaries and the new access control rules introduced in the B-Ambient calculus, and (2) the construction of set $\hat{S}$ of suspect ambients.

We now describe the intuition behind the rule for the *open*-capability. The rule looks for all ambients m labelled ℓ^a with an *open*-capability $\mathbf{open}^{\ell^t}\ n$ on an sibling ambient n labelled $\ell^{a'}$. Then, the result of performing $\mathbf{open}^{\ell^t}\ n$ should also be recorded in either $\hat{I}_B$ or $\hat{I}_E$, depending on the level of protection of the newly generated nestings. This step is split into two distinct cases: (i) the *open* is performed between unprotected (non-boundary) ambients (ii) the *open* is performed between protected ambients/boundaries, requiring that if n is a boundary then also m should be a boundary (because of the access control rule introduced by the B-Ambients semantics). To record the effect of firing the *open*-capability, all the children of the opened ambient n become children of m in $\hat{I}_E$ and $\hat{I}_B$, respectively.

The second part of the rule aims at collecting information about suspect ambients/boundaries: it requires that ambients/boundaries potentially performing an open capability on a suspect ambient/boundary, have to be considered suspect, as well.

The rules for the *in* and *out*-capabilities behave similarly. They mainly differ in the first part of the rule, where $\hat{I}_B$ and $\hat{I}_E$ are constructed. The difference depends on the operational semantics of the B-Ambient calculus: a *in*-capability can be performed independently on the "context" around the performing thread, whereas an *out*-capability is constrained by the access control rule of Figure 2. Moreover, in order to increase the precision of the analysis, by considering the "context" around the ambient performing the capability, i.e., by distinguishing between a protected and an unprotected environment, the rule is split into three sub-cases which correspond to the different scenarios that may happen.

Example 4.2 Let P_4 be the process of Example 4.1. It is possible to prove that the least solution of the analysis for P_4 is the tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$, where $\hat{S} = \{hdata\}$, $\hat{I}_B = \{(b1, h), (b1, b2), (h, c1), (b2, c2)\}$, $\hat{I}_E = \{(env, b1), (env, b2)\}$, and $\hat{H} = \{(b1, container), (h, hdata), (b2, send)\}$. Notice that, with respect to

$\beta^{\mathrm{B}}(P_4)$, the analysis adds in $\hat{I}_E$ the pair $(env, b2)$, representing the possibility for boundary *send* to exit the container. As there is no ambient/boundary performing capabilities over *hdata*, the set $\hat{S}$ of suspect ambients/boundaries is not incremented, and contains *hdata* only.

To see how information about suspect ambients/boudaries helps in detecting indirect information leakage, consider again process P_6 of Example 3.6.

$$P_6 = container^{b1}[\![\ send^{b2}[\![\ \mathbf{in}^{c1}\, hdata.\mathbf{out}^{c2}\, hdata.\mathbf{out}^{c3}\, container\ ]\!]\ \mid \mathbf{open}^{c4}\, download\ ]\!]\,.$$

We proved, in Example 3.9, that P_6 indirectly leaks *hdata*. Consider now the least solution of the analysis for P_6, with $hdata \in \hat{S}$, i.e., the tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$, where $\hat{S} = \{hdata, send\}$, $\hat{I}_B = \{(b1, b2), (b2, c1), (b2, c2), (b2, c3), (b1, c4), \}$, $\hat{I}_E = \{(env, b1), (env, b2)\}$, and $\hat{H} = \{(b1, container), (b2, send)\}$. In this case, *send* is suspect because of the capability performed on *hdata*. Notice also that the analysis reports a potential presence of *send* at the environment level, thus capturing an indirect information leakage. In fact, *send* reaches the environment only after testing the existence of the high level *hdata* ambient. ◇

Example 4.3 We now give an example to motivate why we introduced the concept of suspect ambients in the analysis of [5], instead of starting from the simpler nesting analysis of [28]. The motivation is actually the same that we followed when developing the analysis of [5], i.e., increasing the precision by refining the domain of the analysis from a "flat" set of nestings into the two distinct set of protected ($\hat{I}_B$) and unprotected ($\hat{I}_E$) nestings.

Consider the following process:

$$P_7 = container^{b1}[\![\ test^{b2}[\![\ \mathbf{in}^{c1}\, hdata.\mathbf{out}^{c2}\, hdata.\mathbf{in}^{c3}\, ldata.\mathbf{out}^{c4}\, ldata\ ]\!]\ ]\!] \mid ldata^{l}[\ \mathbf{in}^{c5}\, container\ ]$$

This process is intuitively secure since boundary *test*, which is suspect because of its access to high level data, never exits *container* thus remaining protected in every possible execution. The least solution of the analysis for P_7, with $hdata \in \hat{S}$, can be shown to be the tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$, where $\hat{S} = \{hdata, test\}$, $\hat{I}_B = \{(b1, b2), (b2, c1), (b2, c2), (b2, c3), (b2, c4), (b1, l), (l, c5), (l, b2)\}$, $\hat{I}_E = \{(env, b1), (env, l), (l, c5)\}$, and $\hat{H} = \{(b1, container), (b2, test), (l, ldata)\}$. Analogously to the previous example, *test* is suspect because of the capability performed on *hdata*. Notice also that the analysis reports the potential presence of *test* inside *ldata*, but this happens only after *ldata* enters the *container*, i.e., when *ldata* is protected. In fact, the pair $(l, b2)$ only appears into the set $\hat{I}_B$. From this analysis we can conclude that process P_7 does

not leak *hdata*. In case we would use the simpler nesting analysis of [28], we would obtain all the pairs in $\hat{I}_E \cup \hat{I}_B$ plus new nestings that might result when merging the two sets of nestings. In this particular case, since *ldata* can be at the environment level and *test* can exit from *ldata*, we also obtain that *test* can be at the environment level, i.e., the analysis returns the pair $(env, b2)$, which would erroneously make P_7 appear insecure. The problem is that, when merging the nestings, we loose the information about the fact that $(l, b2)$ only happens when l is protected. This shows that choosing the simpler approach of just one domain for nestings may lead to a strictly less precise analysis. $\diamond$

It is trivial to prove that the set $\{(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \mid \beta^B(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \wedge (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P\}$ of solutions for a process P is a Moore family, i.e., it is closed under greatest lower bound with respect to the ordering $\sqsubseteq$. As a direct consequence, a least solution of the analysis always exists and it may be computed as follows: first apply the representation function to process P, then apply the analysis to validate the correctness of the proposed solution, adding, if needed, new information to the tuple until a fixed-point is reached. More formally, the fixed-point algorithm works as follows:

Algorithm 1 (Fixed-Point Algorithm)

Input: a labelled process P.

(i) Apply the representation function β^B to process P to get a tuple of the form $(\hat{S}^o, \hat{I}^o_B, \hat{I}^o_E, \hat{H})$;
(ii) for all the constraints of the specification of the analysis, validate the tuple $(\hat{S}^i, \hat{I}^i_B, \hat{I}^i_E, \hat{H})$ generated in (i):
  (1) if the constraint is satisfied, continue;
  (2) else, in case the constraint is not satisfied, this is due to the fact that either $\hat{S}^i$ does not consider suspect ambients, or $\hat{I}^i_B$ and $\hat{I}^i_E$ do not consider nestings that may actually occur. In this case, modify $\hat{S}^i$, $\hat{I}^i_B$, or $\hat{I}^i_E$ by adding the "missing" names or pairs, thus getting a new tuple $(\hat{S}^{i+1}, \hat{I}^{i+1}_B, \hat{I}^{i+1}_E, \hat{H})$. Then, go back to (ii) with $i = i + 1$.

The algorithm always terminates, since the number of process labels and names is finite. The complexity is polynomial, as there is at most a quadratic number of nestings, bounding the number of iterations, and each iteration is polynomial in the number of universal quantifications. For more efficient versions of the above algorithm, please refer to [7,31,32].

The correctness of the analysis is proven by showing that every reduction of the semantics is properly mimicked in the analysis. Intuitively, the theorem states that whenever $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ and the representation of P is contained in $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$, we are sure that every ambient that may exercise a capability on a suspect ambient is suspect too and that every nesting of ambients and capabilities in every possible derivative of P is also captured in $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$.

Theorem 4.4 (Subject reduction) *Let P and Q be two processes such that $\beta^{\mathrm{B}}_{\ell,Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \wedge (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P \wedge P \to Q$. Then, $\beta^{\mathrm{B}}_{\ell,Proct}(Q) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \wedge (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$.*

Proof. We proceed by induction on the derivation depth of $P \to Q$.

Base Step: By case analysis on the axioms of Figure 2:

(*InRed*) Let process $P = n^{\ell^a}[\,\mathbf{in}^{\ell^t}\, m.P_1 \mid Q_1\,] \mid m^{\ell^{a'}}[\,R_1\,]$ and process $Q = m^{\ell^{a'}}[\, n^{\ell^a}[\,P_1 \mid Q_1\,] \mid R_1\,]$. Assume that $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \wedge (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$. We obtain three different cases depending on the value of $Proct$ and on the label ℓ^a:

($Proct = \mathrm{True}$) : $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^{a'}), (\ell^{a''}, \ell^a), (\ell^a, \ell^t)\} \subseteq \hat{I}_B$; by $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ (case 1 of the **in** rule) we know that $(\ell^{a'}, \ell^a) \in \hat{I}_B$. By definition, $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(Q) = (s, \{(\ell^{a''}, \ell^{a'}), (\ell^{a'}, \ell^a)\}, \emptyset, \emptyset) \sqcup \beta^{\mathrm{B}}_{\ell^a,Proct}(P_1) \sqcup \beta^{\mathrm{B}}_{\ell^a,Proct}(Q_1) \sqcup \beta^{\mathrm{B}}_{\ell^{a'},Proct}(R_1)$, differing from $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P)$ only on $(\ell^{a'}, \ell^a)$, which we have just proved to belong to $\hat{I}_B$. Thus, we have that $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(Q) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$. Moreover, Q differs from P only in two nestings and in the absence of the **in** capability. Since changes just in the nestings do not have any impact on the analysis (see rule *amb*), then $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ implies $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$.

($Proct = \mathrm{False}, \ell^a \in \mathbf{Lab}^a_B$) : $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^{a'}), (\ell^{a''}, \ell^a)\} \subseteq \hat{I}_E$ and $\{(\ell^a, \ell^t)\} \subseteq \hat{I}_B$; the proof proceeds similarly to the case above by exploiting case 2 of the **in** rule;

($Proct = \mathrm{False}, \ell^a \notin \mathbf{Lab}^a_B$) : $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^{a'}), (\ell^{a''}, \ell^a), (\ell^a, \ell^t)\} \subseteq \hat{I}_E$; the proof proceeds similarly to the cases above by exploiting case 3 of the **in** rule;

(*OutRed*) Let process $P = m^{\ell^{a'}}[\, n^{\ell^a}[\,\mathbf{out}^{\ell^t}\, m.P_1 \mid Q_1\,] \mid R_1\,]$ and process $Q = n^{\ell^a}[\,P_1 \mid Q_1\,] \mid m^{\ell^{a'}}[\,R_1\,]$. We have that $P \to Q$ only if $\ell^a \in \mathbf{Lab}^a_B$ or $\ell^{a'} \notin \mathbf{Lab}^a_B$. Assume that $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \wedge (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$. We obtain three different cases depending on the value of $Proct$ and on the label ℓ^a:

($Proct = \mathrm{True}$) : $\beta^{\mathrm{B}}_{\ell^{a''},Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^{a'}), (\ell^{a'}, \ell^a),$

$(\ell^a, \ell^t)\} \subseteq \hat{I}_B$; by $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ (case 2 of the **out** rule) we know that $(\ell^{a''}, \ell^a) \in \hat{I}_B$. By definition, $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(Q) = (s, \{(\ell^{a''}, \ell^{a'}), (\ell^{a''}, \ell^a)\}, \emptyset, \emptyset) \sqcup \beta^{\mathrm{B}}_{\ell^a, Proct}(P_1) \sqcup \beta^{\mathrm{B}}_{\ell^a, Proct}(Q_1) \sqcup \beta^{\mathrm{B}}_{\ell^{a'}, Proct}(R_1)$, differing from $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P)$ only on $(\ell^{a''}, \ell^a)$ which we have just proved to belong to $\hat{I}_B$. Thus, $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(Q) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$. Moreover, Q differs from P only in two nestings and in the absence of the **out** capability. Since changes just in the nestings do not have any impact on the analysis (see rule *amb*), then $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ implies $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$.

($Proct = \mathrm{False}, \ell^a \in \mathbf{Lab}^a_B$) : $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies that the pair $(\ell^{a''}, \ell^{a'}) \in \hat{I}_E$, $(\ell^{a'}, \ell^a) \in \hat{I}_B \cup \hat{I}_E$ and $(\ell^a, \ell^t) \in \hat{I}_B$; the proof proceeds similarly to the case above by exploiting case 1 of the **out** rule;

($Proct = \mathrm{False}, \ell^a \notin \mathbf{Lab}^a_B$) : by the semantic constraint over boundaries we know that $\ell^a \notin \mathbf{Lab}^a_B$ implies $\ell^{a'} \notin \mathbf{Lab}^a_B$; thus $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^{a'}), (\ell^{a'}, \ell^a), (\ell^a, \ell^t)\} \subseteq \hat{I}_E$; the proof proceeds similarly to the cases above by exploiting case 3 of the **out** rule;

(*OpenRed1*) Let process $P = n^{\ell^a}[\, \mathbf{open}^{\ell^t}\, m.P_1 \mid m^{\ell^{a'}}[\, Q_1 \,] \mid R_1 \,]$ and process $Q = n^{\ell^a}[\, P_1 \mid Q_1 \mid R_1 \,]$. We have that $P \rightarrow Q$ only if $\ell^a \in \mathbf{Lab}^a_B$ or $\ell^{a'} \notin \mathbf{Lab}^a_B$. Assume that $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \ \wedge \ (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$. We obtain three different cases depending on the value of $Proct$ and on the label ℓ^a:

($Proct = \mathrm{True}$) : $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^a), (\ell^a, \ell^{a'}), (\ell^a, \ell^t)\} \subseteq \hat{I}_B$; by $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ (case 2 of the **open** rule) we know that $\{(\ell^a, \ell) | (\ell^{a'}, \ell) \in \hat{I}_B\} \subseteq \hat{I}_B$. By definition, $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(Q) = (s, \{(\ell^{a''}, \ell^a)\}, \emptyset, \emptyset) \sqcup \beta^{\mathrm{B}}_{\ell^a, Proct}(P_1) \sqcup \beta^{\mathrm{B}}_{\ell^a, Proct}(Q_1) \sqcup \beta^{\mathrm{B}}_{\ell^a, Proct}(R_1)$, differing from $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P)$ only on $\beta^{\mathrm{B}}_{\ell^a, Proct}(Q_1)$ in place of $\beta^{\mathrm{B}}_{\ell^{a'}, Proct}(Q_1)$ which is covered by the set $\{(\ell^a, \ell) | (\ell^{a'}, \ell) \in \hat{I}_B\}$ proved to belong to $\hat{I}_B$. Thus, $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(Q) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$. Moreover, Q differs from P only in one nesting and in the absence of the **open** capability. Since changes just in the nestings do not have any impact on the analysis (see rule *amb*), then $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P$ implies $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$.

($Proct = \mathrm{False}, \ell^a \in \mathbf{Lab}^a_B$) : the proof is the same as the case above apart from $(\ell^{a''}, \ell^a)$ belonging to $\hat{I}_E$ instead of $\hat{I}_B$.

($Proct = \mathrm{False}, \ell^a \notin \mathbf{Lab}^a_B$) : by the semantic constraint over boundaries we know that $\ell^a \notin \mathbf{Lab}^a_B$ implies $\ell^{a'} \notin \mathbf{Lab}^a_B$; thus $\beta^{\mathrm{B}}_{\ell^{a''}, Proct}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ implies $\{(\ell^{a''}, \ell^{a'}), (\ell^{a'}, \ell^a), (\ell^a, \ell^t)\} \subseteq \hat{I}_E$; the proof proceeds similarly to the cases above by exploiting case 1 of the **open** rule;

(*OpenRed2*) Let $P = \mathbf{open}^{\ell^t}\, m.P_1 \mid m^{\ell^{a'}}[\, Q_1 \,]$ and $Q = P_1 \mid Q_1$. We have that $P \rightarrow Q$ only if $\ell^{a'} \notin \mathbf{Lab}^a_B$. The proof is analogous to the previous case.

Inductive Step: it is easy by observing that renaming of bound names (α-conversion) and structural congruence do not have any effect on the process representation and on the analysis. In other words, $P =_\alpha Q$ or $P \equiv$

Q imply $\beta^{\mathrm{B}}_{\ell^a,Proct}(P) = \beta^{\mathrm{B}}_{\ell^a,Proct}(Q)$ and $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P$ if and only if $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} Q$. □

4.3 Absence of Indirect Information Leakage

The result of the analysis should be read, as expected, in terms of indirect information flow: no indirect/implicit leakage of information in a process P is possible at run-time if the set of *suspect* ambients remains protected inside security boundaries during the whole execution. The main theorem shows that a process satisfying this condition does not indirectly leak information towards any context that *well-behaves* with respect to the same set of suspect ambients, i.e., contexts which do not introduce flows by themselves.

Before stating the theorem, some additional definitions and a lemma need to be introduced. First, we need to extend the `Protected` predicate of Definition 3.1 to deal with a set of ambient names rather than with the label of a single ambient. Second, we define the set of processes in which the set of suspect ambients remains protected during the run-time execution, and we call them *protective* processes. Finally, we prove that a process which is protective with respect to $\hat{S}$, remains protective when executed inside a context which well-behaves with respect to the same set $\hat{S}$.

Given an analysis $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P$, we write $\texttt{Protected}\ (\hat{S},(\hat{I}_E,\hat{H}))$ to denote that all suspect ambients in $\hat{S}$ are protected with respect to the nestings $\hat{I}_E$ and the labelling $\hat{H}$. Formally, for every $n \in \hat{S}$ and for every label ℓ such that $\exists(\ell,n) \in \hat{H}$, it holds $\texttt{Protected}(\ell,\hat{I}_E)$.

The set $\mathbb{P}_{\hat{S}}$ of all the processes which are protective with respect to a given set of suspect ambients $\hat{S}$, is defined as follows. Notice that the definition can be extended to contexts: a context $\mathcal{C}$ *well-behaves* with respect to $\hat{S}$, written $\mathcal{C} \in \mathbb{C}_{\hat{S}}$ if $\mathcal{C}(\mathbf{0}) \in \mathbb{P}_{\hat{S}}$.

Definition 4.5 (Protective Process) *A process P is* protective *with respect to $\hat{S}$, written*

$P \in \mathbb{P}_{\hat{S}}$ iff
- (i) $\exists \hat{I}_B,\hat{I}_E,\hat{H} : \beta^{\mathrm{B}}(P) \sqsubseteq (\hat{S},\hat{I}_B,\hat{I}_E,\hat{H})$ and $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P$;
- (ii) $\texttt{Protected}\ (\hat{S},(\hat{I}_E,\hat{H}))$.

Given the definition of protective process, the following lemma states that a process which is protective with respect to $\hat{S}$, remains protective when exe-

cuted inside a context which well-behaves with respect to the same set $\hat{S}$.

Lemma 4.6 *Let $P \in \mathbb{P}_{\hat{S}}$ and $\mathcal{C} \in \mathbb{C}_{\hat{S}}$. Then, $\mathcal{C}(P) \in \mathbb{P}_{\hat{S}}$.*

Proof. By Definition 4.5, we need to prove that, let $\beta^{\mathrm{B}}(P) \sqsubseteq (\hat{S}, \hat{I}'_B, \hat{I}'_E, \hat{H}')$, $(\hat{S}, \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} P$ and $\texttt{Protected}\ (\hat{S}, (\hat{I}'_E, \hat{H}'))$, and let $\beta^{\mathrm{B}}(\mathcal{C}(\, test^t[\, \mathbf{0}\,]\,)) \sqsubseteq (\hat{S}, \hat{I}''_B, \hat{I}''_E, \hat{H}'')$, $(\hat{S}, \hat{I}''_B, \hat{I}''_E, \hat{H}'') \models^{\mathrm{B}} \mathcal{C}(\, test^t[\,]\,)$, with $test \notin \hat{S} \cup fn(\mathcal{C})$, and $\texttt{Protected}\ (\hat{S}, (\hat{I}''_E, \hat{H}''))$, then:

(i) $\exists \hat{I}'''_B, \hat{I}'''_E, \hat{H}''' : \beta^{\mathrm{B}}(\mathcal{C}(P)) \sqsubseteq (\hat{S}, \hat{I}'''_B, \hat{I}'''_E, \hat{H}''')$ and $(\hat{S}, \hat{I}'''_B, \hat{I}'''_E, \hat{H}''') \models^{\mathrm{B}} \mathcal{C}(P)$;
(ii) $\texttt{Protected}\ (\hat{S}, (\hat{I}'''_E, \hat{H}'''))$.

In order to build a valid analysis of $\mathcal{C}(P)$ it suffices finding a suitable tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ as an input to the fixed-point algorithm 1. Sets $\hat{I}_B$, $\hat{I}_E$, and $\hat{H}$ can be easily chosen as follows:

- $\hat{I}_B$ can be built by taking the union of sets $\hat{I}'_B$, $\hat{I}''_B$ and $\hat{I}'_E$ (it could be the case that the hole in the context is protected, thus every nesting in P may become protected as well), with all the occurrences of *env* in $\hat{I}'_B$ (or $\hat{I}'_E$) substituted with the label t of ambient *test* (step 1). Then (step 2), *test* ambient, which is only a placeholder of the hole in context $\mathcal{C}$ is passed. Finally (step 3), all the pairs containing the label t of ambient *test* are removed. $\hat{I}_E$ is built similarly; the only difference is in step 1, where only the union of $\hat{I}'_E$ and $\hat{I}''_E$ is taken.
  step 1: $\hat{I}_B = \hat{I}'_B \cup \hat{I}''_B \cup \hat{I}'_E[t/env]$ and $\hat{I}_E = \hat{I}'_E[t/env] \cup \hat{I}''_E$
  step 2: $\hat{I}_B = \hat{I}_B \cup \{(\ell, \ell') \mid (\ell, t), (t, \ell') \in \hat{I}_B\}$ and $\hat{I}_E = \hat{I}_E \cup \{(\ell, \ell') \mid (\ell, t), (t, \ell') \in \hat{I}_E\}$
  step 3: $\hat{I}_B = \hat{I}_B \setminus \{(\ell, \ell') \mid \ell \vee \ell' = t\}$ and $\hat{I}_E = \hat{I}_E \setminus \{(\ell, \ell') \mid \ell \vee \ell' = t\}$
  Sets $\hat{I}_B$ and $\hat{I}_E$ represent all the nesting of process $\mathcal{C}(P)$, in which no interaction between the context $\mathcal{C}$ and P have been considered, i.e., where the execution of $\mathcal{C}$ and P have been considered separately by the two control flow analyses.
- Since labels are supposed to be all different, we can build $\hat{H}$ as the union of $\hat{H}'$ and $\hat{H}''$, i.e., $\hat{H} = \hat{H}' \cup \hat{H}''$.

Starting with $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$, we apply a standard fixed-point algorithm to find a solution of the set of constraints specifying the analysis of process $\mathcal{C}(P)$. The algorithm checks all the constraints and, in case one of them does not hold, it adds either the missing nesting to $\hat{I}_E$ or $\hat{I}_B$, or the name of a suspect ambient to the set $\hat{S}$, or both. The algorithm terminates when a fixed-point is reached, which constitutes a valid analysis that satisfies case (i) above.

We prove, by induction on the algorithm steps, that the algorithm never adds a suspect name to $\hat{S}$, i.e., the solution is of the form $(\hat{S}, \hat{I}^n_B, \hat{I}^n_E, \hat{H})$, with n being the number of algorithm's steps performed to reach the fixed-point, and with $\hat{I}^n_B \supseteq \hat{I}_B$ and $\hat{I}^n_E \supseteq \hat{I}_E$ (that is, $\hat{S}^i = \hat{S}^0$ for all $i \geq 1$). Moreover,

by contradiction, we prove that for any new pair added to $\hat{I}_E$ (new pairs in $\hat{I}_B$ do not change the protectivity of a process) the process remains protective with respect to $\hat{S}$, i.e., $\texttt{Protected}\ (\hat{S}, (\hat{I}_E^n, \hat{H}))$ still holds (case (ii) above).

Base Step ($i = 0$): The only pairs that can violate the Case 2-clause of the *in*, *out*, or *open* rules are the ones that have been added when building $\hat{I}_E$ from $\hat{I}'_E$ and $\hat{I}''_E$ (the same holds for $\hat{I}_B$) which are of the form (ℓ, ℓ') such that $(\ell, t), (t, \ell') \in \hat{I}_E$. Assume that one of these pairs requires a name m to be added to $\hat{S}^0$: this would imply that ℓ' is the label of a capability which has a suspect name as target. Thus, also t should be the label of a suspect ambient. Since t represents *env* in $\hat{I}'_E$, we would have *env* suspect in P, contradicting the fact that $\texttt{Protected}\ (\hat{S}, (\hat{I}'_E, \hat{H}'))$. Thus, no new names are added to $\hat{S}^0$, and $\hat{S}^1 = \hat{S}^0$.

Set $\hat{I}_E$, constructed from $\hat{I}'_E$ and $\hat{I}''_E$, is such that $\texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H}))$. This follows by the construction of $\mathcal{C}(P)$, and by the fact that we have that $\texttt{Protected}\ (\hat{S}, (\hat{I}'_E, \hat{H}'))$ and $\texttt{Protected}\ (\hat{S}, (\hat{I}''_E, \hat{H}''))$.

Inductive Step (i): Assume that at this step the algorithm is adding the pair (ℓ, ℓ') to $\hat{I}^i_E$ (or $\hat{I}^i_B$), i.e., $\hat{I}^{i+1}_E = \hat{I}^i_E \cup \{(\ell, \ell')\}$ (or $\hat{I}^{i+1}_B = \hat{I}^i_B \cup \{(\ell, \ell')\}$). By inductive hypothesis assume also that $\hat{S}^j = \hat{S}^0$ for all $j \leq i$. We consider all the clauses in which the pair (ℓ, ℓ') could have been added:

(in): Case 2 of *in*-rule requires $m \in \hat{S}^{i+1}$ if $\ell \in \mathbf{Lab}^a_H$ (i.e., if the target ambient of the capability is high-level), with $\hat{S}^{i+1} = \hat{S}^i \cup \{m\}$, and m the name of an ambient containing the capability $\mathbf{in}^{\ell t}\ n$. If $\ell \in \mathbf{Lab}^a_H$ then, since $(\ell, n) \in \hat{H}$, $n \in \hat{S}^0$, i.e., ambient n is suspect since the first step of the fixed-point algorithm. In addition, consider the $\mathbf{in}^{\ell t}\ n$ capability inside m:

- either it is inside m since the first step of the algorithm. In this case, m in $\mathcal{C}(P)$ is of the form $m^{\ell'}[\,\mathbf{in}^{\ell t}\ n.P \mid Q\,]$, and m is suspect in $\hat{S}^0$ since the construction of the starting tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$;
- or it has been inherited at one the previous steps of the algorithm (step k, with $0 < k < i$), by the application of the *open*-clause to an ambient x containing the *in*-capability. In this case, m is of the form $m^{\ell'}[\,\mathbf{open}^{\ell'_t}\ x.P \mid x^{\ell''}[\,\mathbf{in}^{\ell t}\ n.Q\,] \mid R\,]$ and x is a suspect ambient, since it contains a capability with a suspect ambient as a target; moreover, $x \in \hat{S}^k$ and $\hat{S}^k = \hat{S}^0$ by inductive hypothesis. Case 2 of the *open*-rule requires $m \in \hat{S}^{k+1}$ if $x \in \hat{S}^k$, and, by inductive hypothesis, we have that $\hat{S}^{k+1} = \hat{S}^0$, i.e., also ambient m is suspect since the first step of the algorithm.

In both cases, we have that $m \in \hat{S}^i$. Thus $\hat{S}^{i+1} = \hat{S}^i$ $(= \hat{S}^0)$.

It remains to show that the process remains protective. Assume by contradiction that the new pair $(\ell\ell')$ makes a suspect ambient become unprotected, that is, $\exists s \in \hat{S}$, with $(\overline{\ell}, s) \in \hat{H}$ s.t. $\texttt{Unprotected}(\overline{\ell}, \hat{I}_E)$, i.e., there exists a path $(env, \ell_1), (\ell_1, \ell_2), \cdots, (\ell, \ell'), \cdots, (\ell_n, \overline{\ell}) \in \hat{I}^{i+1}_E$ such that $\ell_1, \ell_2, \cdots, \ell_n, \ell, \ell' \notin \mathbf{Lab}^a_B$. However, the *in*-rule adds the pair (ℓ, ℓ')

to $\hat{I}_E^i$ only if there exists a label ℓ'' such that:

- (ℓ', ℓ^t), (ℓ'', ℓ'), $(\ell'', \ell) \in \hat{I}_E^i \cup \hat{I}_B^i$ and $\ell' \in \mathbf{Lab}_B^a$, which leads to a contradiction.
- $(\ell'', \ell') \in \hat{I}_E^i$ and $(\ell'', \ell) \in \hat{I}_E^i$. Also in this case we obtain a contradiction, since we would have that s is unprotected also at the previous step because there would have been an unprotected path $(env, \ell_1), (\ell_1, \ell_2), \cdots, (\ell'', \ell'), \cdots, (\ell_n, \overline{\ell}) \in \hat{I}_E^{i+1}$, with $\ell_1, \ell_2, \cdots, \ell_n, \ell, \ell', \ell'' \notin \mathbf{Lab}_B^a$, which leads to a contradiction.

(out): Case 2 of *out*-rule requires $m \in \hat{S}^{i+1}$ if $\ell \in \mathbf{Lab}_H^a$ (i.e., if the target ambient of the capability is high-level), with $\hat{S}^{i+1} = \hat{S}^i \cup \{m\}$, and m the name of an ambient containing the capability $\mathbf{out}^{\ell^t}\, n$. If $\ell \in \mathbf{Lab}_H^a$ then, since $(\ell, n) \in \hat{H}$, $n \in \hat{S}^0$, i.e., ambient n is suspect since the first step of the fixed-point algorithm. In addition, consider the $\mathbf{out}^{\ell^t}\, n$ capability inside m:

- either it is inside m since the first step of the algorithm. In this case m in $\mathcal{C}(P)$ is of the form $m^{\ell'}[\,\mathbf{out}^{\ell^t}\, n.P \mid Q\,]$, and m is suspect in $\hat{S}^0$ as derived when building the starting tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$;
- or it has been inherited at one the previous steps of the algorithm (step k, with $k < i$), by the application of the *open*-clause to an ambient x containing the *out*-capability. In this case, m is of the form $m^{\ell'}[\,\mathbf{open}^{\ell_t'}\, x.P \mid x^{\ell''}[\,\mathbf{out}^{\ell^t}\, n.Q\,] \mid R\,]$ and x is a suspect ambient, since it contains a capability with a suspect ambient as a target; moreover, $x \in \hat{S}^k$ and $\hat{S}^k = \hat{S}^0$ by inductive hypothesis. Case 2 of the *open*-rule requires $m \in \hat{S}^{k+1}$ if $x \in \hat{S}^k$, and, by inductive hypothesis, we have that $\hat{S}^{k+1} = \hat{S}^0$, i.e., also ambient m is suspect since the first step of the algorithm.

In both cases, we have that $m \in \hat{S}^i$, thus $\hat{S}^{i+1} = \hat{S}^i$ $(= \hat{S}^0)$.

It remains to show that the process remains protective. Assume by contradiction that the new pair (ℓ, ℓ') makes a suspect ambient become unprotected, that is, $\exists s \in \hat{S}$, with $(\overline{\ell}, s) \in \hat{H}$ s.t. $\texttt{Unprotected}(\overline{\ell}, \hat{I}_E)$, i.e., there exists a path $(env, \ell_1), (\ell_1, \ell_2), \cdots, (\ell, \ell'), \cdots, (\ell_n, \overline{\ell}) \in \hat{I}_E^{i+1}$ such that $\ell_1, \ell_2, \cdots, \ell_n, \ell, \ell' \notin \mathbf{Lab}_B^a$. However, the *out*-rule adds the pair (ℓ, ℓ') to $\hat{I}_E^i$ only if there exists a label ℓ'' such that:

- $(\ell, \ell'') \in \hat{I}_E^i$, $(\ell'', \ell') \in \hat{I}_E^i \cup \hat{I}_B^i$, and $\ell' \in \mathbf{Lab}_B^a$, which leads to a contradiction.
- $(\ell, \ell'') \in \hat{I}_E^i$ and $(\ell'', \ell') \in \hat{I}_E^i$. Also in this case we obtain a contradiction, since we would have that s is unprotected also at the previous step because there would have been an unprotected path $(env, \ell_1), (\ell_1, \ell_2), \cdots, (\ell, \ell''), (\ell'', \ell'), \cdots, (\ell_n, \overline{\ell}) \in \hat{I}_E^{i+1}$, with $\ell_1, \ell_2, \cdots, \ell_n, \ell, \ell', \ell'' \notin \mathbf{Lab}_B^a$, which leads to a contradiction.

(open): Case 2 of *open*-rule requires $m \in \hat{S}^{i+1}$ if $\ell \in \mathbf{Lab}_H^a$ (i.e., if the target ambient of the capability is high-level), with $\hat{S}^{i+1} = \hat{S}^i \cup \{m\}$, and m the name of an ambient containing the capability $\mathbf{open}^{\ell^t}\, n$. If $\ell \in \mathbf{Lab}_H^a$ then, since $(\ell, n) \in \hat{H}$, $n \in \hat{S}^0$, i.e., ambient n is suspect since the first step of

the fixed-point algorithm. In addition, consider the $\mathbf{open}^{\ell t}\ n$ capability inside m:

- either it is inside m since the first step of the algorithm. In this case, m in $\mathcal{C}(P)$ is of the form $m^{\ell'}[\,\mathbf{open}^{\ell t}\ n.P \mid Q\,]$, and m is suspect in $\hat{S}^0$ as derived when building the starting tuple $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$;
- or it has been inherited at one the previous steps of the algorithm (step k, with $k < i$), by the application of the *open*-clause to an ambient x containing the *open*-capability. In this case, m is of the form $m^{\ell'}[\,\mathbf{open}^{\ell'_t}\ x.P \mid x^{\ell''}[\,\mathbf{open}^{\ell t}\ n.Q\,] \mid R\,]$ and x is a suspect ambient, since it contains a capability with a suspect ambient as a target; moreover, $x \in \hat{S}^k$ and $\hat{S}^k = \hat{S}^0$ by inductive hypothesis. Case 2 of the *open*-rule requires $m \in \hat{S}^{k+1}$ if $x \in \hat{S}^k$, and, by inductive hypothesis, we have that $\hat{S}^{k+1} = \hat{S}^0$, i.e., also ambient m is suspect since the first step of the algorithm.

In both cases, we have that $m \in \hat{S}^i$. Thus $\hat{S}^{i+1} = \hat{S}^i\ (= \hat{S}^0)$.

It remains to show that the process remains protective. Assume by contradiction that the new pair (ℓ, ℓ') makes a suspect ambient become unprotected. In this case, $\exists s \in \hat{S}$, with $(\overline{\ell}, s) \in \hat{H}$ s.t. $\texttt{Unprotected}(\overline{\ell}, \hat{I}_E)$, i.e., there exists a path $(env, \ell_1), (\ell_1, \ell_2), \cdots, (\ell, \ell'), \cdots, (\ell_n, \overline{\ell}) \in \hat{I}_E^{i+1}$ such that $\ell_1, \ell_2, \cdots, \ell_n, \ell, \ell' \notin \mathbf{Lab}_B^a$. The *open*-rule adds the pair (ℓ, ℓ') to $\hat{I}_E^i$ only if there exists a label ℓ'' such that (ℓ, ℓ''), $(\ell'', \ell') \in \hat{I}_E^i$ and $\ell'' \notin \mathbf{Lab}_B^a$. This means that s is unprotected also at the previous step because there would have been an unprotected path $(env, \ell_1), (\ell_1, \ell_2), \cdots, (\ell, \ell''), (\ell'', \ell'), \cdots, (\ell_n, \overline{\ell}) \in \hat{I}_E^{i+1}$, with labels $\ell_1, \ell_2, \cdots, \ell_n, \ell, \ell', \ell'' \notin \mathbf{Lab}_B^a$, thus leading to a contradiction.

The algorithm above always terminates since at most all possible (finite) nestings are added to $\hat{I}_E$ and $\hat{I}_B$. Let n be the number of algorithm's steps performed in order to reach a fixed-point, and let $\hat{I}_B''' = \hat{I}_B^n$, $\hat{I}_E''' = \hat{I}_E^n$, and $\hat{H}''' = \hat{H}$, then we have proved that $\exists \hat{I}_B''', \hat{I}_E''', \hat{H}'''$ such that $(\hat{S}, \hat{I}_B''', \hat{I}_E''', \hat{H}''') \models^{\mathrm{B}} \mathcal{C}(P)$ (the second part of case (i) of the Lemma). Moreover, by $\beta^{\mathrm{B}}(\mathcal{C}(P)) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ (easily proved by the hypotheses $\beta^{\mathrm{B}}(P) \sqsubseteq (\hat{S}, \hat{I}'_B, \hat{I}'_E, \hat{H}')$, $\beta^{\mathrm{B}}(\mathcal{C}(\,test^t[\,]\,)) \sqsubseteq (\hat{S}, \hat{I}''_B, \hat{I}''_E, \hat{H}'')$), and by the fact that $\hat{I}_B^n \supseteq \hat{I}_B$ and $\hat{I}_E^n \supseteq \hat{I}_E$, it follows that $\beta^{\mathrm{B}}(\mathcal{C}(P)) \sqsubseteq (\hat{S}, \hat{I}'''_B, \hat{I}'''_E, \hat{H}'''_E)$ (the first part of case (i) of the Lemma). We have also proved that $\texttt{Protected}\ (\hat{S}, (\hat{I}'''_E, \hat{H}'''))$ still holds (case (ii) of the Lemma).

□

The proof of the main theorem relies on a function, ns, which takes as arguments a process P and a set $\hat{S}$ of ambient/boundary names, and returns process P with all ambients/boundaries belonging to $\hat{S}$, together with the subprocesses they include, syntactically replaced by $\mathbf{0}$. More formally, a *transformation function* $ns(P, \hat{S})$ on process P with suspects names $\hat{S}$ is defined as

follows:

$$ns((\nu n)P, \hat{S}) \equiv \begin{cases} ns(P, \hat{S}) & \text{if } n \in \hat{S} \\ (\nu n)ns(P, \hat{S}) & \text{if } n \notin \hat{S} \end{cases}$$

$$ns(\mathbf{0}, \hat{S}) \equiv \mathbf{0}$$

$$ns(P \mid Q, \hat{S}) \equiv ns(P, \hat{S}) \mid ns(Q, \hat{S})$$

$$ns(!P, \hat{S}) \equiv !ns(P, \hat{S})$$

$$ns(\, n[\, P \,]\, , \hat{S}) \equiv \begin{cases} \mathbf{0} & \text{if } n \in \hat{S} \\ n[\, ns(P, \hat{S}) \,] & \text{if } n \notin \hat{S} \end{cases}$$

$$ns(\mathbf{in}\, n.P, \hat{S}) \equiv \mathbf{in}\, n.ns(P, \hat{S})$$

$$ns(\mathbf{out}\, n.P, \hat{S}) \equiv \mathbf{out}\, n.ns(P, \hat{S})$$

$$ns(\mathbf{open}\, n.P, \hat{S}) \equiv \mathbf{open}\, n.ns(P, \hat{S})$$

The following properties immediately follow from the definition above, and will be useful when proving the main theorem.

Lemma 4.7 *Let $P \in$ **Proc** be of the form $P = n[\, \mathbf{in}\, m.P_1 \mid Q_1 \,] \mid m[\, R_1 \,]$, $n, m \notin \hat{S}$. If $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$, then $P' \equiv (\nu k_1, \cdots, k_z)\, n[\, \mathbf{in}\, m.P_2 \mid Q_2 \,] \mid m[\, R_2 \,] \mid P_3$ with:*

- *$m, n \notin \hat{S}'$, and $k_i \in \hat{S}'$ $\forall i \leq z, z \geq 0$;*
- *$ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}')$, $ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}')$, $ns(R_1, \hat{S}) \equiv ns(R_2, \hat{S}')$, and $ns(P_3, \hat{S}') \equiv \mathbf{0}$.*

Lemma 4.8 *Let $P \in$ **Proc** be of the form $P = m[\, n[\, \mathbf{out}\, m.P_1 \mid Q_1 \,] \mid R_1 \,]$, $m, n \notin \hat{S}$. If $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$, then $P' \equiv (\nu k_1, \cdots, k_z)\, m[\, n[\, \mathbf{out}\, m.P_2 \mid Q_2 \,] \mid R_3 \,] \mid P_3$ with:*

- *$m, n \notin \hat{S}'$, and $k_i \in \hat{S}'$ $\forall i \leq z, z \geq 0$;*
- *$ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}')$, $ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}')$, $ns(R_1, \hat{S}) \equiv ns(R_2, \hat{S}')$, and $ns(P_3, \hat{S}') \equiv \mathbf{0}$.*

Lemma 4.9 *Let $P \in$ **Proc** be of the form $P = n[\, \mathbf{open}\, m.P_1 \mid m[\, Q_1 \,] \mid R_1 \,]$, $m, n \notin \hat{S}$. If $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$, then $P' \equiv (\nu k_1, \cdots, k_z)\, n[\, \mathbf{open}\, m.P_2 \mid \mid m[\, Q_2 \,]\, R_2 \,] \mid P_3$ with:*

- *$m, n \notin \hat{S}'$, and $k_i \in \hat{S}'$ $\forall i \leq z, z \geq 0$;*
- *$ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}')$, $ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}')$, $ns(R_1, \hat{S}) \equiv ns(R_2, \hat{S}')$, and $ns(P_3, \hat{S}') \equiv \mathbf{0}$.*

Lemma 4.10 *Let $P \in$ **Proc** be of the form $P = \mathbf{open}\, m.P_1 \mid m[\, Q_1 \,]$, $m \notin \hat{S}$. If $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$, then $P' \equiv (\nu k_1, \cdots, k_z)\mathbf{open}\, m.P_2 \mid m[\, Q_2 \,] \mid P_3$, with:*

- $m \notin \hat{S}'$, *and* $k_i \in \hat{S}'\ \forall i \leq z, z \geq 0$;
- $ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}')$, $ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}')$, *and* $ns(P_3, \hat{S}') \equiv \mathbf{0}$.

Lemma 4.11 *If* $ns(P, \hat{S}) \equiv \mathbf{0}$, *then* $P \equiv (\nu n_1, \cdots, n_t, m_1, \cdots, m_z)\, n_1[\, P_1\,] \mid \cdots \mid n_k[\, P_k\,]$ *with* $n_i \in \hat{S}\ \forall i \leq k$, $k \geq 0$, $0 \leq t \geq k$, *and* $m_i \in \hat{S}'\ \forall i \leq z, z \geq 0$.

We now show how the previously defined control flow may be applied to prove absence of (indirect) information flow in a given process P: if the set of suspect ambients is protected during the whole run-time execution, we are guaranteed that the system is interference-free.

Theorem 4.12 (Absence of Information Leakage) *If* $P \in \mathbb{P}_{\hat{S}}$, *then,* P *does not leak* $N \subseteq \hat{S}$ *to* $\mathbb{C}_{\hat{S}}$. **Proof.** We have to prove that, if $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P \wedge \texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H}))$, then, for all substitution functions σ_N, we have $P \simeq_{\mathbb{C}_{\hat{S}}} P\sigma_N$, i.e., that P does not leak N to $\mathbb{C}_{\hat{S}}$. To do this, by exploiting Proposition 2.10, we prove $P \approx_{\mathbb{C}_{\hat{S}}} P\sigma_N$. This amounts of proving that, for all well-behaving contexts $\mathcal{C} \in \mathbb{C}_{\hat{S}}$, we have $\mathcal{C}(P) \approx \mathcal{C}(P\sigma_N)$.

In the proof, we proceed as follows:

(1) we define a symmetric relation $\mathcal{S}$, which is proven to be a bisimulation, i.e, we prove that, if $(P, P') \in \mathcal{S}$, then:
  (i) $P \downarrow n$ implies $P' \downarrow n$;
  (ii) $P \longrightarrow Q$ implies that $\exists Q'$ such that $P' \longrightarrow^* Q'$ and $(Q, Q') \in \mathcal{S}$.
(2) we prove that P and $P\sigma_N$ are barbed congruent up to $\mathbb{C}_{\hat{S}}$, i.e., we prove that $(\mathcal{C}(P), \mathcal{C}(P\sigma_N)) \in \mathcal{S}$ for all contexts $\mathcal{C} \in \mathbb{C}_{\hat{S}}$.

Without loss of generality, in the following, we will consider only the case in which P' does not have high names restricted. By using the function ns defined above, we are now able to define a relation $\mathcal{S}$ as follows:

$$\begin{aligned}\mathcal{S} = \{\, (P, P') \mid\ & \exists (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}), (\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \text{ s.t. } \beta^{\mathrm{B}}(P) \sqsubseteq (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}),\\ & \beta^{\mathrm{B}}(P') \sqsubseteq (\hat{S}, \hat{I}'_B, \hat{I}'_E, \hat{H}'), (\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P, (\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} P',\\ & \texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H})), \texttt{Protected}\ (\hat{S}', (\hat{I}'_E, \hat{H}')), ns(P, \hat{S}) \equiv ns(P', \hat{S}')\}.\end{aligned}$$

We now prove that $\mathcal{S}$ is a bisimulation (step 1 of the proof):

(i) By hypothesis $(P, P') \in \mathcal{S}$ and, by definition of $\mathcal{S}$, this means that $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$. The two processes are then structurally equivalent up to suspect names, which are never exhibited since by hypothesis we have $\texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H}))$ and $\texttt{Protected}\ (\hat{S}', (\hat{I}'_E, \hat{H}'))$. Thus, P and P' exhibit the same names.
(ii) The proof is by induction on the depth of the derivation $P \longrightarrow Q$:

Base Step: By case analysis on the axioms of Figure 2.

(*InRed*) Let $P = n[\,\mathbf{in}\, m.P_1 \mid Q_1\,] \mid m[\,R_1\,]$ and $Q = m[\,n[\,P_1 \mid Q_1\,] \mid R_1\,]$. Assume that $(P, P') \in \mathcal{S}$. Then:
- $\exists(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ s.t. $:(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P \wedge \texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H}))$;
- $\exists(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}')$ s.t. $:(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} P' \wedge \texttt{Protected}\ (\hat{S}', (\hat{I}'_E, \hat{H}'))$;
- $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$.

By Theorem 4.4 and the assumptions above, we derive that $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$, and consequently that Q is a protective process. For this reason, $n, m \notin \hat{S}$, then, by Lemma 4.7, $P' \equiv n[\,\mathbf{in}\, m.P_2 \mid Q_2\,] \mid m[\,R_2\,] \mid P_3$, with:
(1) $n, m \notin \hat{S}'$;
(2) $ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}'), ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}'), ns(R_1, \hat{S}) \equiv ns(R_2, \hat{S}')$, and $ns(P_3, \hat{S}') \equiv \mathbf{0}$;
(3) $P_3 = \mathbf{0}$ since we assumed P' to be a protective process.

By applying *(InRed)* to P', we get $Q' \equiv m[\,n[\,P_2 \mid Q_2\,] \mid R_2\,] \mid P_3$, where $P_3 \equiv \mathbf{0}$. By Theorem 4.4, we can derive that $(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} Q'$, and, by (1), (2) and (3) above, we get that $ns(Q, \hat{S}) \equiv ns(Q', \hat{S}')$. By the definition of $\mathcal{S}$, it follows that $(Q, Q') \in \mathcal{S}$.

(*OutRed*) Let $P = m[\,n[\,\mathbf{out}\, m.P_1 \mid Q_1\,] \mid R_1\,]$ and $Q = n[\,P_1 \mid Q_1\,] \mid m[\,R_1\,]$. Assume that $(P, P') \in \mathcal{S}$, then:
- $\exists(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ s.t. $:(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P \wedge \texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H}))$;
- $\exists(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}')$ s.t. $:(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} P' \wedge \texttt{Protected}\ (\hat{S}', (\hat{I}'_E, \hat{H}'))$;
- $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$.

By Theorem 4.4 and the assumptions above, we derive that $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$, and consequently that Q is a protective process. For this reason, $n, m \notin \hat{S}$, then, by Lemma 4.8, $P' \equiv m[\,n[\,\mathbf{out}\, m.P_2 \mid Q_2\,] \mid R_3\,] \mid P_3$, with:
(1) $n, m \notin \hat{S}'$;
(2) $ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}'), ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}'), ns(R_1, \hat{S}) \equiv ns(R_2, \hat{S}')$, and $ns(P_3, \hat{S}') \equiv \mathbf{0}$;
(3) $P_3 = \mathbf{0}$ since we assumed P' to be a protective process.

By applying *(OutRed)* to P', we get $Q' \equiv n[\,P_2 \mid Q_2\,] \mid m[\,R_2\,] \mid P_3$, where $P_3 \equiv \mathbf{0}$. By Theorem 4.4, we can derive that $(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} Q'$, and, by (1), (2) and (3) above, we get that $ns(Q, \hat{S}) \equiv ns(Q', \hat{S}')$. By the definition of $\mathcal{S}$, it follows that $(Q, Q') \in \mathcal{S}$.

(*OpenRed1*) Let $P = n[\,\mathbf{open}\, m.P_1 \mid m[\,Q_1\,] \mid R_1\,]$ and $Q = n[\,P_1 \mid Q_1 \mid R_1\,]$. Let us assume that $(P, P') \in \mathcal{S}$, then:
- $\exists(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ s.t. $:(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P \wedge \texttt{Protected}\ (\hat{S}, (\hat{I}_E, \hat{H}))$;
- $\exists(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}')$ s.t. $:(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} P' \wedge \texttt{Protected}\ (\hat{S}', (\hat{I}'_E, \hat{H}'))$;
- $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$.

By Theorem 4.4 and the assumptions above, we derive that $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$, and consequently that Q is a protective process. For this reason, $m \notin \hat{S}$, then, it follows that: If $n, m \notin \hat{S}$, then, by Lemma 4.9, $P' \equiv n[\,\mathbf{open}\, m.P_2 \mid m[\,Q_2\,] \mid R_2\,] \mid P_3$, with:
(1) $n, m \notin \hat{S}'$;
(2) $ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}'), ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}'), ns(R_1, \hat{S}) \equiv ns(R_2, \hat{S}')$,

and $ns(P_3, \hat{S}') \equiv \mathbf{0}$.

By applying *(OpenRed1)* to P', we get $Q' \equiv m[\, P_2 \mid Q_2 \mid R_2 \,] \mid P_3$, where $P_3 \equiv \mathbf{0}$. By Theorem 4.4, we can derive that $(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} Q'$, and, by (1) and (2) above, we get that $ns(Q, \hat{S}) \equiv ns(Q', \hat{S}')$. By the definition of $\mathcal{S}$, it follows that $(Q, Q') \in \mathcal{S}$.

(*OpenRed2*) Let $P = \mathbf{open}\, m.P_1 \mid m[\, Q_1 \,]$ and $Q = P_1 \mid Q_1$. Assume that $(P, P') \in \mathcal{S}$, then:

- $\exists(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H})$ s.t. $:(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} P \ \wedge$ `Protected` $(\hat{S}, (\hat{I}_E, \hat{H}))$;
- $\exists(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}')$ s.t. $:(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} P' \ \wedge$ `Protected` $(\hat{S}', (\hat{I}'_E, \hat{H}'))$;
- $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$.

By Theorem 4.4 and the assumptions above, we derive that $(\hat{S}, \hat{I}_B, \hat{I}_E, \hat{H}) \models^{\mathrm{B}} Q$, and consequently that Q is a protective process. For this reason, $m \notin \hat{S}$, then, by Lemma 4.10, $P' \equiv \mathbf{open}\, m.P_2 \mid m[\, Q_2 \,] \mid P_3$, with:

(1) $m \notin \hat{S}'$;

(2) $ns(P_1, \hat{S}) \equiv ns(P_2, \hat{S}'), ns(Q_1, \hat{S}) \equiv ns(Q_2, \hat{S}')$, and $ns(P_3, \hat{S}') \equiv \mathbf{0}$;

(3) $P_3 = \mathbf{0}$ since we assumed P' to be a protective process as well.

By applying *(OpenRed2)* to P', we get $Q' \equiv P_2 \mid Q_2$. By Theorem 4.4, we can derive that $(\hat{S}', \hat{I}'_B, \hat{I}'_E, \hat{H}') \models^{\mathrm{B}} Q'$, and, by (1) and (2) above, we get that $ns(Q, \hat{S}) \equiv ns(Q', \hat{S}')$. By the definition of $\mathcal{S}$, it follows that $(Q, Q') \in \mathcal{S}$.

Inductive Step: By case analysis of the last operational rule applied among the ones in Figure 2.

(*ResRed*) Let $P = (\nu n)P_1$, $Q = (\nu n)Q_1$, and $P_1 \longrightarrow Q_1$. Assume $(P, P') \in \mathcal{S}$, i.e., $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$. In this case, P' is of the form $(\nu n)P'_1$ if $n \notin \hat{S}$, or of the form P'_1 if $n \in \hat{S}$, with $ns(P_1, \hat{S}) \equiv ns(P'_1, \hat{S}')$, then $(P_1, P'_1) \in \mathcal{S}$. By induction hypothesis, $\exists Q'_1$ s.t. $P'_1 \longrightarrow^* Q'_1$ and $(Q_1, Q'_1) \in \mathcal{S}$, i.e., $ns(Q_1, \hat{S}) \equiv ns(Q'_1, \hat{S}')$. By applying rule (*ResRed*), P' goes into $Q' = (\nu n)Q'_1$, with $ns(Q, \hat{S}) \equiv ns(Q', \hat{S}')$ since $ns(Q_1, \hat{S}) \equiv ns(Q'_1, \hat{S}')$. It follows that $(Q, Q') \in \mathcal{S}$.

(*AmbRed*) Let $P = n[\, P_1 \,], Q = n[\, Q_1 \,]$, and $P_1 \longrightarrow Q_1$. Assume $(P, P') \in \mathcal{S}$, i.e., $ns(P, \hat{S}) \equiv ns(P', \hat{S}')$. In this case, $n \notin \hat{S}$ since P is assumed to be protective. Then, P' is of the form $P' \equiv n[\, P'_1 \,]$ with $n \notin \hat{S}'$ and $ns(P_1, \hat{S}) \equiv ns(P'_1, \hat{S}')$, and $(P_1, P'_1) \in \mathcal{S}$. By induction hypothesis, $\exists Q'_1$ s.t. $P'_1 \longrightarrow^* Q'_1$ and $(Q_1, Q'_1) \in \mathcal{S}$, i.e., $ns(Q_1, \hat{S}) \equiv ns(Q'_1, \hat{S}')$. By applying rule (*AmbRed*), P' goes into $Q' = (\nu n)Q'_1$, with $ns(Q, \hat{S}) \equiv ns(Q', \hat{S}')$ since $ns(Q_1, \hat{S}) \equiv ns(Q'_1, \hat{S}')$. It follows that $(Q, Q') \in \mathcal{S}$.

Once proved that $\mathcal{S}$ is a barbed bisimulation, by Lemma 4.6 there exists a triplet $(\hat{S}, \hat{I}'', \hat{H} \cup \hat{H}')$ such that $(\hat{S}, \hat{I}'', \hat{H} \cup \hat{H}') \models^{\mathrm{B}} \mathcal{C}(P)$ and `Protected`$(\hat{S}, (\hat{I}'', \hat{H} \cup \hat{H}'))$. Then, it is easy to see that $(\hat{S}\sigma_N, \hat{I}'', \hat{H}\sigma_N \cup \hat{H}') \models^{\mathrm{B}} \mathcal{C}(P\sigma_N)$ and `Protected`$(\hat{S}\sigma_N, (\hat{I}'', \hat{H}\sigma_N \cup \hat{H}'))$ and $ns(\mathcal{C}(P), \hat{S}\sigma_N \cup \hat{S}') = ns(\mathcal{C}(P\sigma_N), \hat{S}\sigma_N \cup \hat{S}')$. Therefore $(\mathcal{C}(P), \mathcal{C}(P\sigma_N)) \in \mathcal{S}$, i.e. they are barbed bisimilar (step 2 of the proof). This concludes the proof. □

Example 4.13 Let us compute the analysis over some of the examples described throughout the chapter. Recall from Example 2.11 that the least analysis for process P_4 returns $\hat{S} = \{hdata\}$, $\hat{I}_B = \{(b1, h), (b1, b2), (h, c1), (b2, c2)\}$, $\hat{I}_E = \{(env, b1), (env, b2)\}$, and $\hat{H} = \{(b1, container), (h, hdata), (b2, send)\}$. Since $\texttt{Protected}\ (h, (\hat{I}_E, \hat{H})) = \text{true}$, P_4 does not leak $\hat{S}$.

Regarding P_6 of Example 3.6, we have that the suspect set $\hat{S}$ computed by the analysis must contain *send* as well, as it performs capabilities over high-level ambients (recall that $send^{b2}[\![\, \mathbf{in}^{c1}\, hdata.\mathbf{out}^{c2}\, hdata.\mathbf{out}^{c3}\, container\,]\!]$). More precisely, the analysis result consists of $\hat{S} = \{hdata, send\}$, $\hat{I}_B = \{(b1, b2), (b2, c1), (b2, c2), (b2, c3), (b1, c4), \}$, $\hat{I}_E = \{(env, b1), (env, b2)\}$, and finally $\hat{H} = \{(b1, container), (b2, send)\}$. In this case, $\texttt{Protected}\ (b2, (\hat{I}_E, \hat{H})) = \text{false}$, thus process P_6 cannot be proved interference-free, and indeed it is not since *send*, a suspect ambient, may exit from *container*. ◇

Notice that in [20] also deadlocks are shown to be potential sources of information leakage. In order to deal with this special kind of information flow, it is sufficient to strength Definition 3.8 by using an equivalence notion which is deadlock-suspect, such as bisimilarity. Observe that Theorem 4.12 still holds even with this stronger notion of information leakage.

5 Related Work and Conclusion

The main novelty of the approach presented in this paper is that we face the problem of detecting at the same time both direct and indirect information leakage (non-interference) in the context of Mobile Ambients. In [4], we described a tool that implements the analysis for direct information leakage detection in the core Mobile Ambient calculus, and in [7] we proved that the complexity of this analysis keeps reasonable (polynomial both in space and in time). We expect similar experimental results also in the case of the analysis of indirect information leakage detection, as it can be seen as a more sophisticated algorithm on a richer language.

The most related contributions in the area mainly focused on either extending the ambient calculus thus enhancing its expressive power, or on building suitable type systems or control flow analyses to verify security properties.

Among the type systems approaches, it is worth to mention [11], where the authors introduce a new type system for tracking the behaviour of mobile computations. Using groups, the type system can impose to an ambient behavioural constraints on the set of ambients it may cross and the set of ambients it may open. It has the effect of statically preventing certain communications through a mandatory access control policy, and can block accidental or ma-

licious leakage of secrets. Dezani and Salvo [15] extend the work of Cardelli et al. just mentioned, with a type system that also expresses security levels associated with ambients and provide further control over ambient movement and opening.

Among the control flow analysis approaches, a lot of effort has been done to enrich the abstract domains in order to obtain more detailed analyses that could be used to verify different security properties. In [16], the author proposes an analysis on a non-standard semantics of mobile ambients with first order communications in order to distinguish different recursive instances of threads, and to identify which threads can be launched in which ambients and which ambient names can be communicated to which threads. In [30,28], an exponential analysis for computing occurrences of threads inside ambients is described. In [25], the authors propose an abstract interpretation framework for MA based on a normal semantics. Unfortunately, the more detailed are the abstract domains of the analysis, the more complex is the analysis. In addition, the fact that a different semantics is used, make it more difficult to compare the approach with the type system ones. In any case, our approach to address indirect information leakage can be easily integrated within all of these analyses.

In the context of language extensions, some valuable proposals are described in [8–10,14,26]. *Safe Ambients* is a modification of Mobile Ambients, where a movement or an ambient dissolution can take place only when the affected ambient agrees, offering the corresponding coaction. *Boxed Ambients* [10] is another variant of the Ambient calculus with a completely different model of communication, which results from dropping the *open* capability. In their paper, Bugliesi et al. define also a type system that provides an effective mechanism for resource protection and access control. In [24], Degano et al. present a control flow analysis that mainly focuses on access control. The analysis relies on the use of coactions as a filter to control access to resources.

As already said in the introduction, as far as we know the only work towards the study of non-interference in the context of Mobile Ambients is [13], where a type system that guarantees that well-typed programs do not interfere when in parallel with any high-level source is studied for Boxed Ambients. One of the priorities as future work will be to carefully compare these two approaches. This may also give interesting insights on the trade-off between accuracy and efficiency and usability between type system and control-flow analysis techniques. Their results indeed rely critically on the choice of contextual equivalence as the underlying equivalence relation, thus do not extend to finer equivalence relationships, such as barbed congruence.

References

[1] M. Abadi and A. D. Gordon. A Bisimulation Method for Cryptographic Protocols. *Nordic Journal of Computing*, 5(4):267–303, Winter 1998.

[2] D. E. Bell and L. J. La Padula. Secure Computer Systems: Unified Exposition and Multics Interpretation. ESD-TR-75-306, MITRE MTR-2997, MITRE Corporation, Mar. 1976.

[3] C. Braghin. *Static Analysis of Security Properties in Mobile Ambients*. PhD thesis, Università Ca' Foscari di Venezia, 2005.

[4] C. Braghin, A. Cortesi, S. Filippone, R. Focardi, F. L. Luccio, and C. Piazza. Banana: A Tool for Boundary Ambients Nesting ANAlysis. In H. Garavel and J. Hatcliff, editors, *Proc. of 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 437–441. Springer–Verlag, Berlin, 2003.

[5] C. Braghin, A. Cortesi, and R. Focardi. Security Boundaries in Mobile Ambients. *Computer Languages*, 28(1):101–127, Nov 2002.

[6] C. Braghin, A. Cortesi, and R. Focardi. Information Leakage Detection in Boundary Ambients. In *Proc. of Computing: The Australasian Theory Symposium (CATS'03)*, volume 78 of *Electronic Notes on Theoretical Computer Science*, pages 119–139. Elsevier Science Inc., New York, 2003.

[7] C. Braghin, A. Cortesi, F. L. Luccio, R. Focardi, and C. Piazza. Nesting Analysis of Mobile Ambients. *Computer Languages, Systems and Structures*, 30(3–4):207–230, Oct-Dec 2004.

[8] M. Bugliesi and G. Castagna. Secure Safe Ambients. In *POPL'01 Proc. 28th ACM Symposium on Principles of Programming Languages*, pages 222–235. ACM Press, New York, 2001.

[9] M. Bugliesi and G. Castagna. Behavioral Typing for Safe Ambients. *Computer Languages*, 28(1):61–99, Nov. 2002. Revised and extended version of [8].

[10] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Proc. of 4th Int. Conference on Theoretical, Aspects of Computer Science (TACS'01)*, number 2215 in Lecture Notes in Computer Science, pages 38–63. Springer–Verlag, Berlin, 2001.

[11] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient Groups and Mobility Types. In *Proc. of IFIP International Conference on Theoretical Computer Science (TCS'00)*, volume 1872 of *Lecture Notes in Computer Science*, pages 333–347. Springer–Verlag, Berlin, 2000.

[12] L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[13] S. Crafa, M. Bugliesi, and G. Castagna. Information Flow Security for Boxed Ambients. In *Proc. of Int. Workshop on Foundations of Wide Area Networks(F-WAN'02)*, volume 66(3) of *Electronic Notes on Theoretical Computer Science*, 2002.

[14] P. Degano, F. Levi, and C. Bodei. Safe Ambients: Control Flow Analysis and Security. In J. He and M. Sato, editors, *Proc. of the 6th Asian Computing Science Conference (ASIAN'00)*, volume 1961 of *Lecture Notes in Computer Science*, pages 199–214. Springer–Verlag, Berlin, Dec. 2000.

[15] M. Dezani-Ciancaglini and I. Salvo. Security Types for Mobile Safe Ambients. In J. He and M. Sato, editors, *Proc. of 6th Asian Computing Science Conference (ASIAN'00)*, volume 1961 of *Lecture Notes in Computer Science*, pages 215–236. Springer–Verlag, Berlin, 2000.

[16] J. Feret. Abstract Interpretation-Based Static Analysis of Mobile Ambients. In *Eighth International Static Analysis Symposium (SAS'01)*, number 2126 in LNCS. Springer-Verlag, 2001. Springer-Verlag.

[17] R. Focardi. *Analysis and Automatic Detection of Information Flows in Systems and Networks*. UBLCS-99-16, University of Bologna, July 1999.

[18] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[19] R. Focardi and R. Gorrieri. The Compositional Security Checker: A Tool for the Verification of Information Flow Security Properties. *IEEE Transactions on Software Engineering*, 23(9):550–571, Sept. 1997.

[20] R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design - Tutorial Lectures*, volume 2171 of *Lecture Notes in Computer Science*. Springer–Verlag, Berlin, 2001.

[21] R. Focardi, R. Gorrieri, and F. Martinelli. Information Flow Analysis in a Discrete-Time Process Algebra. In P. Syverson, editor, *Proc. of 13th Computer Security Foundations Workshop (CSFW'00)*, pages 170–184. IEEE Computer Society Press, 2000.

[22] J. A. Goguen and J. Meseguer. Security Policies and Security Models. In *Proc. of IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society Press, Apr. 1982.

[23] A. D. Gordon and L. Cardelli. Equational Properties of Mobile Ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.

[24] F. Levi and C. Bodei. Security Analysis for Mobile Ambients. In *Proc. of the Workshop on Issues on the Theory of Security (WITS'00)*, 2000.

[25] F. Levi and S. Maffeis. On Abstract Interpretation of Mobile Ambients. *Information and Computation*, 188:179–240, January 2004.

[26] F. Levi and D. Sangiorgi. Mobile Safe Ambients. *ACM Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[27] J. Morris. *Lambda-Calculus Models of Programming Languages.* PhD thesis, MIT, 1968.

[28] F. Nielson, R. R. Hansen, and H. R. Nielson. Abstract Interpretation of Mobile Ambients. *Science of Computer Programming, Special Issue on Static Analysis*, 47(2–3):145–175, 2003.

[29] F. Nielson, H. R. Nielson, and C. Hankin. *Principles of Program Analysis.* Springer–Verlag, Berlin, 1999.

[30] F. Nielson, H. R. Nielson, R. R. Hansen, and J. Jensen. Validating Firewalls in Mobile Ambients. In *Proc. of 10th Int. Conference on Concurrency Theory (CONCUR99)*, volume 1664 of *Lecture Notes in Computer Science*, pages 463–477. Springer–Verlag, Berlin, 1999.

[31] F. Nielson, H. R. Nielson, and H. Seidl. Automatic Complexity Analysis. In D. L. Metayer, editor, *Proc. of the 10th European Symposium On Programming (ESOP'02)*, volume 2305 of *Lecture Notes in Computer Science*, pages 243–261. Springer–Verlag, Berlin, 2002.

[32] F. Nielson and H. Seidl. Control-Flow Analysis in Cubic Time. In D. Sands, editor, *Proc. of 10th European Symposium On Programming (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 252–268. Springer–Verlag, Berlin, 2001.

[33] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients: An abstraction for biological compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.

P, Q	$::=$	$(\nu n)P$	restriction
	$\mid$	$\mathbf{0}$	inactivity
	$\mid$	$P \mid Q$	composition
	$\mid$	$!P$	replication
	$\mid$	$n[\,P\,]$	ambient or boundary
	$\mid$	$\mathbf{in}\, n.P$	capability to enter n
	$\mid$	$\mathbf{out}\, n.P$	capability to exit n
	$\mid$	$\mathbf{open}\, n.P$	capability to open n

Fig. 1. Syntax of Boundary Ambients.

(InRed)	$n[\,\mathbf{in}\, m.P \mid Q\,] \mid m[\,R\,] \rightarrow m[\, n[\,P \mid Q\,] \mid R\,]$
(OutRed)	$m[\, n[\,\mathbf{out}\ m.P \mid Q\,] \mid R\,] \longrightarrow n[\,P \mid Q\,] \mid m[\,R\,]$
	$m \in \mathbf{Bound} \Rightarrow n \in \mathbf{Bound}$
(OpenRed)	$n[\,\mathbf{open}\ m.P \mid m[\,Q\,] \mid R\,] \longrightarrow n[\,P \mid Q \mid R\,]$
	$m \in \mathbf{Bound} \Rightarrow n \in \mathbf{Bound}$
	$\mathbf{open}\ m.P \mid m[\,Q\,] \longrightarrow P \mid Q \qquad m \in \mathbf{Amb}$
(ResRed)	$P \rightarrow Q \Rightarrow (\nu n)P \rightarrow (\nu n)Q$
(AmbRed)	$P \rightarrow Q \Rightarrow n[\,P\,] \rightarrow n[\,Q\,]$
(CompRed)	$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$
($\equiv$ Red)	$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$

Fig. 2. Reduction Rules: $P \rightarrow Q$.

$$
\begin{array}{ll}
P \equiv P & P \mid Q \equiv Q \mid P \\
P \equiv Q \Rightarrow Q \equiv P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
P \equiv Q, Q \equiv R \Rightarrow P \equiv R & !P \equiv P \mid !P \\
P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q & (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \\
P \equiv Q \Rightarrow P \mid R \equiv Q \mid R & (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \quad \text{if } n \notin fn(P) \\
P \equiv Q \Rightarrow !P \equiv !Q & (\nu n)\, m[\,P\,] \;\equiv\; m[\,(\nu n)P\,] \quad \text{if } n \neq m \\
P \equiv Q \Rightarrow\; n[\,P\,] \;\equiv\; n[\,Q\,] & P \mid \mathbf{0} \equiv P \\
P \equiv Q \Rightarrow \mathbf{in}\; n.P \equiv \mathbf{in}\; n.Q & (\nu n)\mathbf{0} \equiv \mathbf{0} \\
P \equiv Q \Rightarrow \mathbf{out}\; n.P \equiv \mathbf{out}\; n.Q & !\mathbf{0} \equiv \mathbf{0} \\
P \equiv Q \Rightarrow \mathbf{open}\; n.P \equiv \mathbf{open}\; n.Q &
\end{array}
$$

Fig. 3. Structural Congruence: $P \equiv Q$.

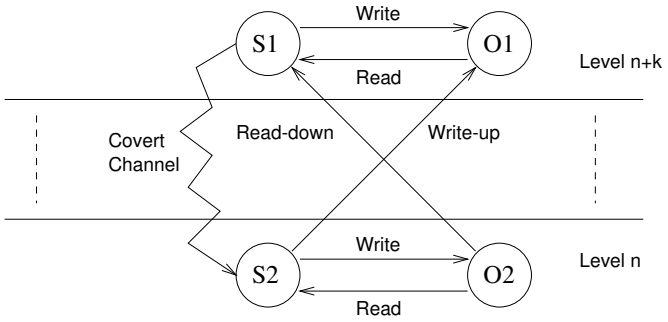


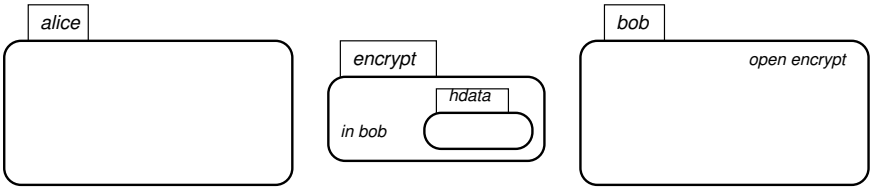
Fig. 4. Information flows in the BLP model [17].

$$
\begin{array}{lll}
(res) & Nest_\ell((\nu n)P) & = Nest_\ell(P) \\
(zero) & Nest_\ell(\mathbf{0}) & = \emptyset \\
(par) & Nest_\ell(P \mid Q) & = Nest_\ell(P) \cup Nest_\ell(Q) \\
(repl) & Nest_\ell(!P) & = Nest_\ell(P) \\
(amb) & Nest_\ell(\, n^{\ell^a}[\,P\,]\,) & = Nest_{\ell^a}(P) \cup \{(\ell, \ell^a)\} \\
(in) & Nest_\ell(\mathbf{in}^{\ell^t}\; n.P) & = Nest_\ell(P) \cup \{(\ell, \ell^t)\} \\
(out) & Nest_\ell(\mathbf{out}^{\ell^t}\; n.P) & = Nest_\ell(P) \cup \{(\ell, \ell^t)\} \\
(open) & Nest_\ell(\mathbf{open}^{\ell^t}\; n.P) & = Nest_\ell(P) \cup \{(\ell, \ell^t)\}
\end{array}
$$

Fig. 5. Definition of Function $Nest$.

(a) *alice* needs to send confidential data *data* to *bob*.

(b) the confidential data is encrypted with a shared key.



(c) *data* is sent encrypted over the communication channel.

(d) *hdata* is safely received by *bob*.

(e) *bob* accesses confidential data *hdata* by decrypting it.

Fig. 6. *alice* and *bob* exchanging confidential information.

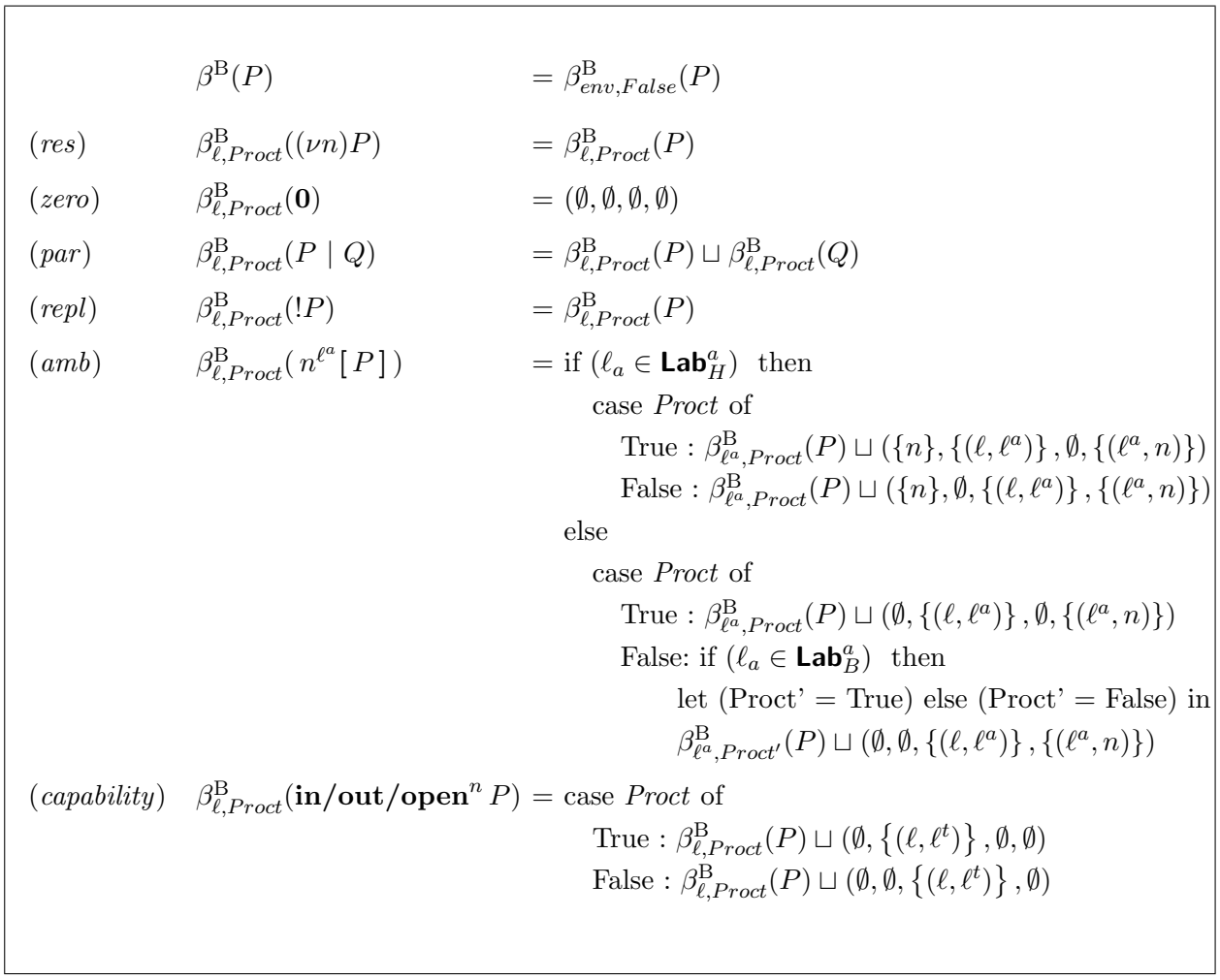


Fig. 7. Representation Function for the Control Flow Analysis.

(res) $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} (\nu n)P$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P$

$(zero)$ $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} \mathbf{0}$ always

(par) $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P \mid Q$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P \ \wedge\ (\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} Q$

$(repl)$ $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} !P$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P$

(amb) $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} n^{\ell^a}[\,P\,]$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P$

$(\mathbf{in})$ $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} \mathbf{in}^{\ell^t} n.P$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P \wedge$

$\forall \ell^a,\ell^{a'},\ell^{a''} \in \mathbf{Lab}^a$ s.t. $(\ell^{a'},n) \in \hat{H}$:

1: case $(\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_B$

$\Longrightarrow\ (\ell^{a'},\ell^a) \in \hat{I}_B$

case $(\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ \ell^a \in \mathbf{Lab}^a_B$

$\Longrightarrow$ if $(\ \ell^{a'} \in \mathbf{Lab}^a_B)$ then $(\ell^{a'},\ell^a) \in \hat{I}_B$

else $(\ell^{a'},\ell^a) \in \hat{I}_E$

case $(\ell^a,\ell^t) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ \ell^a \notin \mathbf{Lab}^a_B$

$\Longrightarrow$ if $(\ \ell^{a'} \in \mathbf{Lab}^a_B)$ then $(\ell^{a'},\ell^a) \in \hat{I}_B \ \wedge\ \left\{(\ell,\ell') \in \hat{I}_E \mid path_E(\ell^a,\ell)\right\} \subseteq \hat{I}_B$

else $(\ell^{a'},\ell^a) \in \hat{I}_E$

2: $\forall m \in \lfloor \mathbf{Names} \rfloor$ s.t. $(\ell^a,m) \in \hat{H}$:

$((\ell^a,\ell^t) \in \hat{I}_B \cup \hat{I}_E \ \wedge\ n \in \hat{S}) \implies m \in \hat{S}$,

$(\mathbf{out})$ $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} \mathbf{out}^{\ell^t} n.P$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P \wedge$

$\forall \ell^a,\ell^{a'},\ell^{a''} \in \mathbf{Lab}^a$ s.t. $(\ell^{a'},n) \in \hat{H} \ \wedge\ (\ell^{a'} \notin \mathbf{Lab}^a_B \vee \ell^a \in \mathbf{Lab}^a_B)$:

1: case $(\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I}_E \cup \hat{I}_B \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ \ell^a \in \mathbf{Lab}^a_B$

$\Longrightarrow\ (\ell^{a''},\ell^a) \in \hat{I}_E$

case $(\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I}_B \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_B$

$\Longrightarrow\ (\ell^{a''},\ell^a) \in \hat{I}_B$

case $(\ell^a,\ell^t) \in \hat{I}_E \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I}_E \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}_E \ \wedge\ \ell^a \notin \mathbf{Lab}^a_B$

$\Longrightarrow\ (\ell^{a''},\ell^a) \in \hat{I}_E$

2: $\forall m \in \lfloor \mathbf{Names} \rfloor$ s.t. $(\ell^a,m) \in \hat{H}$:

$((\ell^a,\ell^t) \in \hat{I}_B \cup \hat{I}_E \ \wedge\ n \in \hat{S}) \implies m \in \hat{S}$,

$(\mathbf{open})$ $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} \mathbf{open}^{\ell^t} n.P$ iff $(\hat{S},\hat{I}_B,\hat{I}_E,\hat{H}) \models^{\mathrm{B}} P \wedge$

$\forall \ell^a,\ell^{a'} \in \mathbf{Lab}^a$ s.t. $(\ell^{a'},n) \in \hat{H} \ \wedge\ (\ell^{a'} \notin \mathbf{Lab}^a_B \vee \ell^a \in \mathbf{Lab}^a_B)$:

1: case $(\ell^a,\ell^t) \in \hat{I}_E \ \wedge\ (\ell^a,\ell^{a'}) \in \hat{I}_E \ \wedge\ \ell^a \notin \mathbf{Lab}^a_B$

$\Longrightarrow\ \left\{(\ell^a,\ell) \mid (\ell^{a'},\ell) \in \hat{I}_E\right\} \subseteq \hat{I}_E$

case $(\ell^a,\ell^t) \in \hat{I}_B \ \wedge\ (\ell^a,\ell^{a'}) \in \hat{I}_B$

$\Longrightarrow\ \left\{(\ell^a,\ell) \mid (\ell^{a'},\ell) \in \hat{I}_B\right\} \subseteq \hat{I}_B$

2: $\forall m \in \lfloor \mathbf{Names} \rfloor$ s.t. $(\ell^a,m) \in \hat{H}$:

$((\ell^a,\ell^t) \in \hat{I}_B \cup \hat{I}_E \ \wedge\ n \in \hat{S}) \implies m \in \hat{S}$,

Fig. 8. Specification of the Control Flow Analysis.