Flow policy awareness for distributed information flow security

Ana Almeida Matos[a,b], Jan Cederquist[a,b]

[a] *Instituto de Telecomunicações (SQIG), Lisbon, Portugal*
[b] *Universidade de Lisboa (IST), Lisbon, Portugal*

Abstract

In the context of distributed and mobile computing, information flow security must deal with the decentralized nature of security policies. This issue is particularly challenging when migrating programs are given the flexibility to perform declassifying operations, as these might be acceptable or not depending on the thread's current computation domain.

This paper studies the compliance of programs consisting of one or more migrating threads to flow polices that can change dynamically via declassification declarations and program migration. It exploits the principle of separating the enabling and controlling dimensions of declassification. At the programming language level, declassification is enabled by assuming a purely declarative declassification construct, and its compliance to the relevant allowed flow policies is facilitated by considering a flow policy context testing construct that provides the programmer with flow policy awareness. Then, two security properties that articulate the compliance of program behavior to relevant information flow policies are studied:

1. *Non-disclosure for networks*, which requires the compliance of information leaks that are encoded by programs to the declared declassifications.
2. *Flow policy confinement* requires that declassifications that are used by programs respect the allowed flow policy of the context in which they execute.

We present mechanisms that are based on type and effect systems, ranging from purely static mechanisms to hybrid combinations with dynamic features, for enforcing the above properties on an expressive distributed higher-order lambda-calculus with imperative features and code migration.

Keywords: information flow, declassification, distribution, code mobility

1. Introduction

The rapid evolution and increasing pervasiveness of web technologies in sensitive aspects of daily life presses the problem of web security as an urgent concern. Indeed, the new possibilities offered by the the ubiquitous and globally connected nature of the Internet can just as well be exploited by parties with hazardous intentions, with an unprecedented potential for damage. Web programming has emerged as a new and distinct paradigm in itself, subject to new and specific forms of security vulnerabilities. Two specific traits of web applications are their distributed nature, as they are executed in different physical locations, and their plasticity which is based on the assemblage and execution of code from different origins, known as mobile code. In this paper we contribute to the young and quickly developing research topic of language-based web security [1] by considering the most basic foundations of the Web as a distributed network with code mobility.

Research in language based security has placed a lot of attention on the study of information flow properties and enforcement mechanisms [36]. Information flow security regards the control of how dependencies between information of different security levels can lead to information leakage during program execution. This kind of program behavior can be controlled using *information flow* analyses [36, 9], by detecting dependencies in programs that could encode flows of information from private to publicly available resources. Information flow properties range in strictness from pure absence of information leaks, classically known as non-interference [21], to more flexible properties that allow for *declassification* to take place in a controlled manner [38].

So far, most studies of declassification have been directed towards local computation scenarios, thus overlooking decentralization issues that are inherent to distributed settings. Indeed, enforcement of confidentiality in networks must deal with distributed security policies, since different *computation domains* (or *sites*) follow different security orientations. For example, migrating programs that were conceived to comply to certain flow policies don't necessarily respect those of the computational locations they might end up executing at. This problem seems to be beyond the grasp of single declassification constructs that can restrict by whom, when, what, or where in the program declassification can be performed [38], since now the question is: *in which context?*

Some security minded distributed network models have been proposed with the purpose of controlling the migration of code between computation sites, such as by means of programmable domains [11], type systems [30], or boundary transposition control that performs automatic checks to incoming code [22]. It appears natural to apply migration control techniques to the problem of controlling declassification by preventing programs from migrating to sites if they would potentially violate that site's flow policies. However, we fall short on technical mechanisms that would allow, on one hand, for a site to know what are the most flexible flow policies that a program sets up for its own executions; on another hand, for a program to know how flexible is the flow policy of the context in which it is running.

In this paper we show that the issue of enabling and controlling flexible information flow policies in computations that can spread out over locations that are governed by different security requirements, can be addressed at the programming language level. We propose to remove some of the burden of restricting declassification away from the declassification instruction itself, and transfer it to new program constructs that provide awareness about the flow policy of the context in which it is running. Programmers can then be given the power to predict alternatives to the pieces of code that contain the forbidden declassification operations they would wish to use. This opens the possibility to write programs that can safely run under any flow policy. Furthermore, it becomes acceptable to rely on mechanisms that reject the execution of programs that, being unaware of the flow policy of the context, blindly encode disallowed declassifications.

Separating the problems of enabling and of controlling flexible information flow policies paves the way to a modular composition of security properties that can be studied independently. Here we treat the former as an information flow control problem, in the frame of the *non-disclosure* property[3], while the latter is isolated as the problem of ensuring that declassifications that are performed by mobile code comply to the flow policy that is allowed at the computation domain where they are performed. We refer to this new property as *flow policy confinement*, and enforce it using migration control techniques. This paper addresses the technical problem of how to build suitable enforcement mechanisms that enable domains to check incoming code against their own allowed flow policies.

Intuitions. The *non-disclosure property* is a generalization of non-interference. It uses information provided by the program semantics describing which flow policies are valid at different points of the computation, to ensure that, at each step, all information flows comply with the valid flow policy. In order to enable local dynamic changes to the valid flow policy, the programming language may be enriched with a *flow declaration* construct (flow F in M) that simply declares the flow policy F as valid in its scope M. It is then easy to set up more flexible flow policy environments for delimited blocks of code, as for instance the part of a program that is executed by authenticated users:

$$(\text{if } \textit{authenticated} \text{ then } (\text{flow } F_{permissive} \text{ in } M) \text{ else } N)$$

This program declares that flows in M conform to a policy that is extended by $F_{permissive}$. In other words, M may contain declassifications that comply to $F_{permissive}$.

Once the language is enriched with flow declarations (or any other means for expressing declassification), some mechanism for controlling its usage is desirable. This is particularly relevant in mobile code settings. For instance, a computation domain d might want to impose a limit to the flexibility of the flow declarations that it executes, and prevent incoming code from containing:

$$(\text{flow } F_{all\_is\_allowed} \text{ in } M)$$

In the above example, the flow declaration validates any information flow that might occur in M, regardless of what is considered acceptable by d. This motivates the notion of a domain's *allowed flow policy*, which represents the flow

policy that should rule for all programs that are running at a certain domain. We can then define the notion of *confinement with respect to a flow policy* as a property of programs that can only perform steps that comply with that allowed flow policy. We will see that this property can be formalized by making use of the information about the declared flow policies that is provided by the semantics.

At the moment that a program is written, it might be hard to anticipate which flow policies will be imposed at execution time by the domains where the program will run. In a distributed context with code mobility, the problem becomes more acute, since the computation site might change *during* execution, along with the allowed flow policy with which the program must comply. In order to provide programs with some awareness regarding the flow policy that is ruling in the current computation domain, we introduce the *allowed-condition*, written (allowed F then M else N), that tests whether the flow policy F is allowed by the current domain and executes branches M or N accordingly. Programs can then offer alternative behaviors to be taken in case the domains they end up at do not allow declassifications of the kind they wished to perform:

$$(\text{allowed } F_{disclose\_secret} \text{ then } M \text{ else } plan\_B)$$

The allowed-condition brings no guarantees that the "*plan_B*" of the above program does not disclose just as much as the M branch. However, misbehaving programs can be rejected by the domains where they would like to execute, so its chances of being allowed to execute are increased by adequately "protecting" portions of code containing declassifications by appropriate allowed-conditions.

Domains can statically check incoming code against their own flow policies, ideally assisted by certificates that are carried by the program, and then decide upon whether those programs should be "let in". A certificate could consist of information about all the flow policies that are declared in the program and do *not* appear within the "allowed" branch of an allowed-condition. We call this flow policy the *declassification effect* of the program, and provide a type system for obtaining it. Then, while the program

$$(\text{allowed } F_1 \text{ then } M \text{ else } (\text{flow } F_2 \text{ in } N))$$

would have a declassification effect that includes F_2 – meaning that it should only be allowed to run in domains where F_2 is allowed –, the program

$$(\text{allowed } F \text{ then } (\text{flow } F \text{ in } M) \text{ else } N)$$

(assuming that M and N have no flow declarations) would have an empty declassification effect – meaning that it could be safely allowed to run in any domain.

Furthermore, when programs consist of more than one thread running concurrently, the same program might need to comply to more than one allowed flow policy simultaneously.

An illustrative scenario could be that of a set of personal mobile appliances, such as smart-phones. Due to their inter-connectivity (web, Bluetooth), they form networks of highly responsive computing devices with relatively limited resources, and that handle sensitive information (personal location, contacts, passwords). This combination demands for scalable and efficient mechanisms for

ensuring privacy in a distributed setting with code mobility. From an abstract perspective, each device forms a *computation domain* with specific capabilities and restrictions, and in particular information flow policies for protecting data and other computing *threads* that are running concurrently in the same domain. We refer to these policies as the *allowed flow policy* of the domain. Flow policy confinement ensures that domains do not execute code that might perform declassifications that break their own allowed policies.

Let us consider, for example, an application for supporting two users (Alice and Bob) in choosing the best meeting point and path for reaching each other by means of public transportation. In order to produce advice that takes into account the current context (recent user locations, traffic conditions, weather) threads containing code for building updated travel maps are downloaded by Alice and Bob at runtime (their travel). The recommended path and meeting point can be improved by deducing the users' personal preferences from data that it collects from the mobile devices (e.g. content of stored images, file types). Users might, however, have privacy restrictions regarding that data, in the form of allowed flow policies that the downloaded threads must comply to. The following naive program creates a thread for gathering data that helps select the meeting point. Since the meeting point will necessarily be revealed to Bob, this part of the program should only allowed to run if it respects which private information Alice accepts to leak to Bob.

```
 1 newthread {                       // Creates thread at Alice's device
 2    ref zoo=0; ref bookstore=0;    // to choose between zoo or bookstore
 3    allowed                        // If allowed by Alice's policy
 4       (L_IMGS < L_BOB /\          // to leak image contents
 5        L_FILES < L_BOB)           // and file types to Bob
 6       flow (L_IMGS < L_BOB /\     //   Declares a declassification
 7             L_FILES < L_BOB)      //   with same policy
 8          processImgs(zoo);        //     weighs images with animals
 9          searchFiles(bookstore); //     weighs e-book files
10          if (zoo > bookstore)    //     inspects sensitive data...
11             meetAt(ZOO);         //       and influences meeting point
12             meetAt(BOOKSTORE);
13    meetAt(random);               // If not allowed, uses other criteria
14 } at D_ALICE
```

As the above code is deployed, device `D_ALICE` must decide whether it is safe to execute the thread or not. Clearly, the decision must be taken quickly so as to not disrupt the purpose of the application. Ultimately, it is based on an analysis of the code, giving special attention to the points where declassifications occur.

Contributions. This paper presents a simple language-based framework for studying information flow security in distributed security settings in the presence of code mobility. In this model, computation domains are units of *allowed flow policies*, which have a scope that is local to each domain. While the formulation of the security properties is largely language-independent, a concrete language is defined and considered for the purpose of examples and as a target to the proposed enforcement mechanisms. It consists of an expressive distributed

higher-order imperative lambda-calculus with remote thread creation. The latter language feature implies in particular that programs might need to comply to more than one dynamically changing allowed flow policy simultaneously. The main technical contributions are:

1. A new programming construct (allowed F then M else N) that tests the flexibility of the allowed flow policy imposed by the domain where it is currently located and can act accordingly.
2. A refinement of the Non-disclosure for Networks property [4] that is more suitable for settings where migration is subjective.
3. A new security property, named *Flow Policy Confinement* that regards the compliance of declassification operations that are performed by programs will to the valid allowed flow policies where they take place.
4. A comparative study of three enforcement mechanisms for flow policy confinement in the form of migration control mechanisms for deciding whether or not certain programs should be allowed to execute at each site. These are based on a type and effect system, and differ on the emphasis that is placed on static and runtime effort:
   (a) A purely static type and effect system for enforcing flow policy confinement.
   (b) A type and effect system for checking migrating threads at runtime, that is more precise than the one in point 4a.
   (c) A static-time informative pre-processing type and effect system for annotating programs with a *declassification effect*, for a more efficient and precise mechanism than the one in point 4b.

This paper revises, unifies and expands work that is presented in the conference articles [2], [5], and part of [6]. Some proofs are omitted for space reasons, but are available in the Appendix.

Outline of the paper. We start by defining the formal security and language setting of the paper (Section 2). Two main sections follow, dedicated to the security analyses of Non-disclosure for Networks (Section 3) and Flow Policy Confinement (Section 4). In each, the formal properties are proposed (Subsections 3.1 and 4.1), and enforcement mechanisms are presented (Subsections 3.2 and 4.2 to 4.4), and their soundness is proved. In the latter, a closer look at the enforcement of Flow Policy Confinement by means of migration control mechanisms. We study three type and effect-based mechanisms for enforcing confinement that place different weight over static and run time, and draw conclusions regarding their efficiency and precision. Finally we discuss related work (Section 5) and conclude (Section 6).

2. Setting

2.1. Security Setting

The study of confidentiality traditionally relies on a lattice of security levels [18], corresponding to security clearances, that is associated to information

containers in the programming language. The idea is that information pertaining to references labeled with l_2 can be legally transferred to references labeled with l_1 only if l_1 is at least as confidential as l_2. *Flow policies* can then be seen as a means for relaxing the basic security lattice, by establishing additional legal flow directions between security levels. Formally, such flow policies consist of downward closure operators that collapse security levels of a basic security lattice into lower ones [7].

2.1.1. Abstract requirements

Basic lattice of security levels. Security levels $l, j, k \in \boldsymbol{Lev}$ can be seen as representing read-access rights, and can be ordered according to their confidentiality by means of a relation $\sqsubseteq$, where $l_1 \sqsubseteq l_2$ means that l_2 is at least as confidential as l_1. It can also be seen as a flow relation, since information can flow from l_1 to l_2 without becoming accessible to any additional reader.

In this paper security levels are assumed to form an abstract lattice $\mathcal{L} = \langle \boldsymbol{Lev}, \sqsubseteq, \sqcap, \sqcup, \top, \bot \rangle$, where: the meet operation $\sqcap$ gives, for any two security levels l_1, l_2, the most confidential security level that allows for all the readers of levels l_1 and l_2; the join operation $\curlyvee$ gives, for any two security levels l_1, l_2, the least confidential security level that only allows for readers of both levels l_1 and l_2; the most confidential security level $\top$ does not allow any reader; and the least confidential security level $\bot$ allows all readers.

Lattice of flow policies. Flow policies $A, F \in \boldsymbol{Flo}$ can be ordered according to their permissiveness by means of a *permissiveness relation* $\preccurlyeq$, where $F_1 \preccurlyeq F_2$ means that F_1 is at least as permissive as F_2.

We assume an abstract lattice of flow policies that supports a pseudo-subtraction operation $\langle \boldsymbol{Flo}, \preccurlyeq, \curlywedge, \curlyvee, \mho, \Omega, \backsim \rangle$, where: the meet operation $\curlywedge$ gives, for any two flow policies F_1, F_2, the strictest policy that allows for both F_1 and F_2; the join operation $\curlyvee$ gives, for any two flow policies F_1, F_2, the most permissive policy that only allows what both F_1 and F_2 allow; the most restrictive flow policy $\mho$ does not allow any information flows; and the most permissive flow policy Ω that allows all information flows. Finally, the pseudo-subtraction operation $\backsim$ between two flow policies F_1 and F_2 [1] represents the most permissive policy that allows everything that is allowed by the first (F_1), while excluding all that is allowed by the second (F_2); it is defined as the relative pseudo-complement of F_2 with respect to F_1, i.e. the greatest F such that $F \curlywedge F_2 \preccurlyeq F_1$.

Relaxed lattice of security levels. The base pre-lattice of security levels can be relaxed by means of flow policies [7], by extending the permissivity of the flow relation $\sqsubseteq$. The new more general flow relation $\sqsubseteq^F$ that is determined by the flow policy F now enables the information flows that are allowed by F. For

[1] This operation is used for refining the static analysis of the policy-testing construct, and is not a requirement of the security properties that are studied here.

a given flow policy F, security levels now form a lattice $\mathcal{L}(F) = \langle F(\boldsymbol{Lev}), \sqsubseteq^F, \sqcap^F, \sqcup^F, \top^F, \bot^F \rangle$.

2.1.2. Concrete example

Considering a concrete security setting that meets the abstract requirements defined above can provide helpful intuitions. In the remainder of this paper we use in examples the case of flow policies that operate over the security lattice where security levels are sets of principals $p, q \in \boldsymbol{Pri}$.

Basic lattice of security levels. Security levels consist of sets of principals $l \subseteq \boldsymbol{Pri}$, similar to read-access lists. In this setting, security levels are ordered by means of the flow relation $\supseteq$.

Lattice of flow policies. Flow policies then consist of binary relations on $\boldsymbol{Pri}$, which can be understood as representing additional directions in which information is allowed to flow between principals: a pair $(p, q) \in F$, most often written $p \prec q$, is to be understood as "information may flow from p to q". New more permissive security lattices are obtained by collapsing security levels into possibly lower ones, by closing them with respect to the valid flow policy. Writing $F_1 \preccurlyeq F_2$ means that F_1 allows flows between at least as many pairs of principals as F_2. The relation is here defined as $F_1 \preccurlyeq F_2$ iff $F_2 \subseteq F_1^*$ (where F^* denotes the reflexive and transitive closure of F): The meet operation is then defined as $\curlywedge = \cup$, the join operation is defined as $F_1 \curlyvee F_2 = F_1^* \cap F_2^*$, the top flow policy is given by $\mho = \emptyset$, the bottom flow policy is given by $\Omega = \boldsymbol{Pri} \times \boldsymbol{Pri}$, and the pseudo-subtraction operation is given by $\smile\!\!= -$ (set subtraction).

Relaxed lattice of security levels. In order to define $\sqsubseteq^F$ we use the notion of *F-upward closure* of a security level l, defined as $l \uparrow_F = \{q \mid \exists p \in l.\ p\ F^*\ q\}$. The F-upward closure of l contains all the principals that are allowed by the policy F to read information labeled l. A more permissive flow relation can now be derived as follows [32, 3]:

$$l_1 \sqsubseteq^F l_2 \stackrel{\text{def}}{\Leftrightarrow} \forall q \in l_2\ .\ \exists p \in l_1\ :\ p\ F^*\ q \ \Leftrightarrow\ (l_1 \uparrow_F) \supseteq (l_2 \uparrow_F)$$

Since in a pre-lattice the meet and join operations are not unique, here we chose $l_1 \sqcap l_2 = l_1 \cup l_2$ and $l_1 \sqcup l_2 = (l_1 \uparrow_F) \cap (l_2 \uparrow_F)$. Consequently, we have $\top = \emptyset$ and $\bot = \boldsymbol{Pri}$.

Notice that $\sqsubseteq^F$ extends $\supseteq$ in the sense that $\sqsubseteq^F$ is larger than $\supseteq$ and that $\sqsubseteq^\emptyset = \supseteq$. In other words, for the base security lattice (where the flow policy parameter is $\mho$), the flow relation coincides with reverse inclusion of security levels, while the join operator is simply given by $\sqcup = \cap$.

2.2. Language Setting

We now present the basic language requirements to which the technical developments of this paper apply. We then define a concrete instance of the language that suits these requirements. It will be used for providing illustrative examples, and as a target for the enforcement mechanisms.

Security Levels	$l, j \in \textbf{\textit{Lev}}$	*Reference Names*	$a, b \in \textbf{\textit{Ref}}$	*Values*	$V \in \textbf{\textit{Val}}$
Flow Policies	$A, F \in \textbf{\textit{Flo}}$	*Thread Names*	$m, n \in \textbf{\textit{Nam}}$	*Expressions*	$M, N \in \textbf{\textit{Exp}}$
Types	$\tau, \sigma, \theta \in \textbf{\textit{Typ}}$	*Domain Names*	$d \in \textbf{\textit{Dom}}$		

Figure 1: Syntax of Basic Elements of the Language

2.2.1. Abstract requirements

Networks are flat juxtapositions of domains, each containing a store and a pool of threads, which are subjected to the local allowed flow policy of the domain. Information is associated to references in $\textbf{\textit{Ref}}$, and can be seen as information containers to which values of the language pertaining to a given type in $\textbf{\textit{Typ}}$ can be assigned, and to thread names in $\textbf{\textit{Nam}}$ that are mapped into different locations when a distributed setting is considered. The basic elements of the language, summarized in Figure 1, are thus references, domains, and threads, whose names are drawn from disjoint countable sets $a, b \in \textbf{\textit{Ref}}$, $d \in \textbf{\textit{Dom}} \neq \emptyset$ and $m, n \in \textbf{\textit{Nam}}$, respectively.

As mentioned earlier, security levels are associated to information holders by means of labelings. We define a *reference labeling* $\Sigma : \textbf{\textit{Ref}} \to \textbf{\textit{Lev}} \times \textbf{\textit{Typ}}$, whose left projection corresponds to the usual security labeling $\Sigma_1 : \textbf{\textit{Ref}} \to \textbf{\textit{Lev}}$ that assigns security levels to references, and right projection corresponds to the *type labeling* $\Sigma_2 : \textbf{\textit{Ref}} \to \textbf{\textit{Typ}}$ that determines the type of values that can be assigned to each reference. We also define a *thread labeling* $\Upsilon : \textbf{\textit{Nam}} \to \textbf{\textit{Lev}}$, that assigns security levels to thread names. This mapping is used in Subsection 3.1. In this paper, the mapping between reference and thread names and their corresponding security annotations and types might be informally denoted as subscript of names in the context of examples.

Threads run concurrently in *pools* $P : \textbf{\textit{Nam}} \to \textbf{\textit{Exp}}$, which are mappings from thread names to expressions (denoted as sets of threads). *Stores* $S : \textbf{\textit{Ref}} \to \textbf{\textit{Val}}$ map reference names to values. *Position-trackers* $T : \textbf{\textit{Nam}} \to \textbf{\textit{Dom}}$, map thread names to domain names, and are used to keep track of the locations of threads in the network. The pool P containing all the threads in the network, the position tracker T that keeps track of their positions, and the store S containing all the references in the network, form *configurations* $\langle P, T, S \rangle$, over which the evaluation relation is defined in the next subsection. We refer to the pairs $\langle S, T \rangle$ as *states*, and pairs $\langle P, T \rangle$ as *thread configurations*. The flow policies that are allowed by each domain are kept by the *allowed-policy mapping* $W : \textbf{\textit{Dom}} \to \textbf{\textit{Flo}}$ from domain names to flow policies.

The '$W \vdash^{\Sigma,\Upsilon}$' turnstile gives a security context to the definition of the semantics, making explicit the allowed flow policy of each domain in the network, and the valid reference and thread labelings. These parameters are fixed, and can be omitted when clear from the context. The transitions $\xrightarrow[F]{d}$ are decorated with the name of the domain d where each step is taking place and the flow policy F declared by the evaluation context where they are performed. The

$$
\begin{array}{lll}
\textit{Values} & V & ::= () \mid x \mid a \mid (\lambda x.M) \mid tt \mid ff \\
\textit{Pseudo-values} & X & ::= V \mid (\varrho x.X) \\
\textit{Expressions} & M, N & ::= X \mid (M\ N) \mid (M; N) \mid (\mathrm{ref}_{l,\theta}\ M) \mid (!\ N) \mid (M := N) \mid \\
 & & \quad (\text{if } M \text{ then } N_t \text{ else } N_f) \mid (\text{flow } F \text{ in } M) \mid \\
 & & \quad (\text{allowed } F \text{ then } N_t \text{ else } N_f) \mid (\mathrm{thread}_l\ M \text{ at } d)
\end{array}
$$

Figure 2: Syntax of Expressions

semantics does not depend on this information, which is used for the purpose of the security analysis. The relation $\twoheadrightarrow$ denotes the reflexive closure of the transition relation $\xrightarrow[F]{d}$.

2.2.2. Concrete object language

The distributed language that we use is a an imperative higher-order λ-calculus with reference creation, where we include a flow policy declaration construct (for directly manipulating flow policies [3] and the new flow policy tester construct that branches according to whether a certain flow policy is allowed in the program's computing context, obtained by adding a notion of computing domain, to which we associate an allowed flow policy, and a code migration primitive. Threads are also named in order to keep track of their position in the network. Programs executing in different domains are subjected to different allowed flow policies – this is what distinguishes local computations from global computations, and is the main novelty in this language. We opt for a rather simplistic memory model, assuming memory to be shared by all programs and every computation domain, in a transparent form. Nevertheless, as we will see in Section 3, while this allows us to avoid synchronization issues that are not central to this work, we do not avoid the distributed nature of the model.

Syntax. The syntax of expressions defined in Figure 2 is based on an imperative higher order λ-calculus that includes boolean values, recursion (provided by the $(\varrho x.X)$ construct), conditional branching, reference and remote thread creation, declassification and a context-policy testing construct.

Reference names are not associated to any security levels or types at the language level (in this aspect we depart from [2]), and make use of the reference labelings defined in Subsection 2.2.1 only during the security analysis. Nevertheless, reference names can be created at runtime, by a construct that is annotated with a type and security level that should be associated with the new reference.

The new features are the flow declaration and the allowed-condition. The flow declaration construct is written (flow F in M), where M is executed in the context of the current flow policy *extended with* F; after termination the

current flow policy is restored, that is, the scope of F is M. The allowed-condition is similar to a standard boolean condition, with the difference that in (allowed F then N_t else N_f) the branches N_t or N_f are executed according to whether or not F is allowed by the site's allowed flow policy.

The remote thread creator ($\text{thread}_l\ M$ at d) spawns a new thread with security level and expression M in domain d, to be executed concurrently with other threads at that domain. It functions as a migration construct when the new domain of the created thread is different from that of the parent thread. The security level l is the confidentiality level that is associated to the knowledge of the position of the thread in the network.

Example. The allowed flow policy A of a site restricts the flow policies that can be set up by programs running in that site. Then, the (allowed F then M else N) construct tests whether F is allowed by A, and can safely set up a flow declaration for F in its "allowed" branch. A typical usage of the construct could then be

$$\begin{array}{lll}(\text{allowed } F_{H\prec L} & \text{then} & (\text{flow } F_{H\prec L} \text{ in } (x_L := (!\ y_H))) \\ & \text{else} & \textit{plan\_B})\end{array} \tag{1}$$

where the flow the flow policy $F_{H\prec L}$ represents a flow policy that allows information to flow from level H to level L.

Operational semantics. In order to define the operational semantics, expressions are represented using *evaluation contexts*, which specify a call-by-value evaluation order:

$$\begin{array}{c}\mathrm{E} ::= [] \mid (\mathrm{E}\ N) \mid (V\ \mathrm{E}) \mid (\mathrm{E}; N) \mid (\text{ref}_{l,\theta}\ \mathrm{E}) \mid (!\ \mathrm{E}) \\ (\mathrm{E} := N) \mid (V := \mathrm{E}) \mid (\text{if E then } N_t \text{ else } N_f)\end{array}$$

We write $\mathrm{E}[M]$ to denote an expression where the sub-expression M is placed in the evaluation context E, obtained by replacing the occurrence of [] in E by M. The flow policy that is permitted by the evaluation context E is denoted by $\lceil \mathrm{E} \rceil$. It consists of a lower bound (see Section 2) to all the flow policies that are declared by the context:

$$\begin{array}{cc}\lceil [] \rceil = \mho, & \lceil (\text{flow } F \text{ in E}) \rceil = F \curlywedge \lceil \mathrm{E} \rceil, \\ \lceil \mathrm{E}'[\mathrm{E}] \rceil = \lceil \mathrm{E} \rceil, & \textit{when } \mathrm{E}' \textit{ has no flow declarations}\end{array}$$

The following basic notations and conventions are used for defining transitions. For a mapping Z, we define $\text{dom}(Z)$ as the domain of a given mapping Z. We say a name is fresh in Z if it does not occur in $\text{dom}(Z)$. We denote by $\text{rn}(P)$ and $\text{dn}(P)$ the set of reference and domain names, respectively, that occur in the expressions of P. We let $\text{fv}(M)$ be the set of variables occurring free in M. We restrict our attention to well formed configurations $\langle P, T, S\rangle$ satisfying the conditions that $\text{rn}(P) \subseteq \text{dom}(S)$, that $\text{dn}(P) \subseteq \text{dom}(W)$, that $\text{dom}(P) \subseteq \text{dom}(T)$, and that, for every $a \in \text{dom}(S)$, $\text{rn}(S(a)) \subseteq \text{dom}(S)$ and $\text{dn}(S(a)) \subseteq \text{dom}(W)$. We denote by $\{x \mapsto W\}M$ the capture-avoiding substitution of W for the free occurrences of x in M. The operation of adding or updating the image of an object z to z' in a mapping Z is denoted $[z := z']Z$.

The small step operational semantics of the language is defined in Figure 3. The last rule establishes that the execution of a pool of threads is compositional

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[((\lambda x.M)\ V)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[\{x \mapsto V\}M]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\text{if } tt \text{ then } N_t \text{ else } N_f)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[N_t]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \mathrm{E}[(\text{if } \mathit{ff} \text{ then } N_t \text{ else } N_f)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[N_f]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(V; N)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[N]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\varrho x.X)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[(\{x \mapsto (\varrho x.X)\}\ X)]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\text{flow } F \text{ in } V)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[V]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(!\ a)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[S(a)]^m\}, T, S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(a := V)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[()]^m\}, T, [a := V]S\rangle$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\mathrm{ref}_{l,\theta}\ V)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[a]^m\}, T, [a := V]S\rangle, \text{ where } a \text{ fresh in } S \text{ and } \Sigma(a) = \theta$$

$$\frac{W(T(m)) \preccurlyeq F}{W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\text{allowed } F \text{ then } N_t \text{ else } N_f)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[N_t]^m\}, T, S\rangle}$$

$$\frac{W(T(m)) \not\preccurlyeq F}{W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\text{allowed } F \text{ then } N_t \text{ else } N_f)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[N_f]^m\}, T, S\rangle}$$

$$W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\mathrm{thread}_l\ N \text{ at } d)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[()]^m, N^n\}, [n := d]T, S\rangle, \text{ where } n \text{ fresh in } T \text{ and } \Sigma(a) = \theta$$

$$\frac{W \vdash^{\Sigma,\Upsilon} \langle P, T, S\rangle \xrightarrow[F]{d} \langle P', T', S'\rangle \quad \langle P \cup Q, T, S\rangle \text{ is well formed}}{W \vdash^{\Sigma,\Upsilon} \langle P \cup Q, T, S\rangle \xrightarrow[F]{d} \langle P' \cup Q, T', S'\rangle}$$

Figure 3: Operational Semantics

(up to the choice of new names). Notice that W, representing the allowed flow policies associated to each domain, is never changed. For simplicity, memory is assumed to be shared by all programs and every computation domain, in a transparent form.

Thread names are used in two situations: When a new thread is created, its new fresh name is added to the position-tracker, associated to the parameter domain. As with reference creation, the security level that is associated to the new thread does not influence the semantics, but is used later by the security analysis. Thread names are also used when an allowed-condition is performed: the tested flow policy is compared to the allowed flow policy of the site where that particular thread is executing.

One can prove that the semantics preserves well-formedness of configurations, and that a configuration with a single thread has at most one transition,

up to the choice of new names.

According to the chosen semantics, dereferencing and assigning to remote references can be done transparently. One may wonder whether it is correct to consider a system with a shared global state as distributed. We point out that in this model, the flow policies are distributed, while the behavior of a program fragment may differ on different machines. As an example, consider the thread

$$(\text{allowed } F \text{ then } (y_L := 1) \text{ else } (y_L := 2))^m \tag{2}$$

running in a network such that $W(d_1) = F_1$ and $W(d_2) = F_2$, where $W(d_1) \preccurlyeq F$ but $W(d_2) \not\preccurlyeq F$. The thread will perform different assignments depending on whether $T(m) = d_1$ or $T(m) = d_2$. In Section 3 we will see that their behavior is distinguishable by an information flow bisimulation relation. In other words, the network does exhibit a distributed behavior.

3. Controlling Information Flow

In this section we revisit the problem of controlling information flow in the context of a distributed setting with code mobility. We present a new refined formalization of the *Non-disclosure for Networks* property, and argue for its suitability. Then, we analyze the behavior of the new allowed condition construct in light of the new property. We then present a type and effect system for enforcing the property over the concrete language of Subsection 2.2 and establish soundness of the enforcement mechanism.

3.1. Non-disclosure for Networks

Non-disclosure states that, at each computation step performed by a program, information flows respects the flow policy that is declared by the evaluation context where the step is performed. The property is naturally defined for sets of concurrent threads in terms of an information-flow bisimulation [37, 40, 12, 19], that, at each execution point, relates the outcomes of each possible step that is performed over states (stores) that coincide on their "low" region, where the notion of "low" is customized with the currently declared flow policy [3]. It can be generalized to a distributed setting by extending the notion of state in order to include position trackers [4].

Low equality. As we will see towards the end of this section, the position of a thread in the network can reveal information about the values in the memory. For this reason, we must use a notion of low-equality that is extended to states. Furthermore, in order to keep track of the security levels that are associated to threads, the relation must be parameterized by the thread mapping Υ (see Subsubsection 2.2.1). Given a security labeling Σ_1, two memories S_1 and S_2 are said to be indistinguishable at level $l \in L$ with respect to a flow policy F, written $S_1 =^{\Sigma_1}_{F,l} S_2$, if they coincide in all references assigned to security levels less or equal than l.

Definition 3.1 (Low-Equality)**.** *The low-equality between states* $\langle T_1, S_1\rangle$ *and* $\langle T_2, S_2\rangle$ *with respect to a flow policy* F *and a security level* l*, written* $\langle T_1, S_1\rangle =^{\Sigma_1,\Upsilon}_{F,l} \langle T_2, S_2\rangle$*, if and only if for every reference name* $a \in \textbf{\textit{Ref}}$*, if* $\Sigma_1(a) \sqsubseteq^F l$*, then* $S_1(a) = S_2(a)$ *holds, and for any thread name* $n \in \textbf{\textit{Nam}}$*, if* $\Upsilon(n) \sqsubseteq^F l$*, then* $T_1(n) = T_2(n)$ *holds.*

This relation is transitive, reflexive and symmetric.

Store compatibility. The language defined in Section 2.2 is a higher-order language, where values stored in memory can be used by programs to build expressions that are then executed. For example, the expression $((!\ a)\ ())$ can evolve into an insecure program when running on a memory that maps a reference a to a lambda-abstraction whose body consists of an insecure expression. In order to avoid considering all such programs insecure, it is necessary to make assumptions concerning the contents of the memory. Here, memories are assumed to be compatible to the given security setting and typing environment, requiring typability of their contents with respect to the type system that is defined in the next subsection (see Definition 3.5), and can be shown to be preserved by the semantics.

Definition on thread configurations. Intuitively, if a program is shown to be related to itself by means of an information flow bisimulation, one can conclude that it has the same behavior regardless of changes in the high part of the memory. In other words, the high part of the memory has not interfered with the low part, i.e., no security leak has occurred. A secure program can then be defined as one that is related to itself by the above bisimulation.

When imposing restrictions on the behaviors of bisimilar pools of threads, the definition in [4] resets the state arbitrarily at each step of the bisimulation game. This accounts for changes that might be induced by threads that are external to the pools under consideration, thus enabling compositionality of the property. However, in a context where migration is subjective (i.e. only the thread itself can trigger its own migration), resetting the position tracker arbitrarily is unnecessary. In fact, it leads to a property that is overly restrictive. In the following program M_{insec} can be a direct leak that is not placed within a flow declaration:

$$(\text{thread}_l\ (\text{allowed } F \text{ then } () \text{ else } M_{insec}) \text{ at } d) \tag{3}$$

The above program is intuitively secure (regarding Non-disclosure for Networks) if $W(d) \preccurlyeq F$ and insecure otherwise, as the body of the thread is known to be executed at domain d. However, it is considered insecure by the original definition, as if the allowed condition is executed over "fresh" thread configurations such that the thread is located at a domain where F is *not* allowed, then the branch with the illegal expressions M_{insec} would be executed. It is then reasonable to relax the power of the attacker, by focusing on the behavior of threads when coupled with their possible locations on the network. The following information-flow bisimulation fixes the position tracker across the bisimulation steps:

Definition 3.2 ($\approx^{\Sigma,\Upsilon}_{\Gamma,l}$)**.** *Consider an allowed-policy mapping W, a reference labeling Σ, and a typing environment Γ. A $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation is a symmetric relation $\mathcal{R}$ on thread configurations that satisfies, for all P_1,T_1,P_2,T_2, and (W,Σ,Γ)-compatible stores S_1,S_2:*

$$\langle P_1,T_1\rangle\ \mathcal{R}\ \langle P_2,T_2\rangle \text{ and } W\vdash\langle P_1,T_1,S_1\rangle \xrightarrow{d}_{\boldsymbol{F}} \langle P_1',T_1',S_1'\rangle$$
$$\text{and } \langle T_1,S_1\rangle =^{\Sigma,\Upsilon}_{\boldsymbol{F},l} \langle T_2,S_2\rangle$$

with $(\mathrm{dom}(S_1')-\mathrm{dom}(S_1))\cap\mathrm{dom}(S_2)=\emptyset$ *and* $(\mathrm{dom}(T_1')-\mathrm{dom}(T_1))\cap\mathrm{dom}(T_2)=\emptyset$ *implies that there exist P_2',T_2',S_2' s.t.:*

$$W\vdash\langle P_2,T_2,S_2\rangle \twoheadrightarrow \langle P_2',T_2',S_2'\rangle \text{ and } \langle T_1',S_1'\rangle =^{\Sigma,\Upsilon}_{\mho,l} \langle T_2',S_2'\rangle$$
$$\text{and } \langle P_1',T_1'\rangle\ \mathcal{R}\ \langle P_2',T_2'\rangle$$

Furthermore, S_1',S_2' are still (W,Σ,Γ)-compatible. The largest $(W,\Sigma,\Upsilon,\Gamma,l)$-bisimulation, the union of all such bisimulations, is denoted $\approx^{\Sigma,\Upsilon,}_{\Gamma,l}$.

For any Σ, Υ, Γ and l, the set of pairs of thread configurations where threads are values is an $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation. Furthermore, the union of a family of $(\Sigma,\Upsilon,\Gamma,l)$-bisimulations is a $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation. Consequently, $\approx^{\Sigma,\Upsilon,}_{\Gamma,l}$ exists.

Despite the technical difference regarding the new focus on thread configurations, the intuitions that explain the above definition are maintained. We briefly recall them here, and refer the reader to [4] for more explanations. The reason why the above bisimulation potentially relates more programs than one for Non-interference is the stronger premise $\langle T_1,S_1\rangle =^{\Sigma_1}_{F,l} \langle T_2,S_2\rangle$, which assumes pairs of states that coincide "to a greater extent", thus "facilitating" the reproduction of the behavior by the opposite thread configuration. The absence of a condition on the flow policy of the matching move for P_2 enables all expressions without side-effects (such as) to be bisimilar, independently of the flow policy that is declared by their evaluation contexts. Clearly, the relation $\approx^{\Sigma}_{\Gamma,l}$ is not reflexive since, as only "secure" programs, as defined next, are bisimilar to themselves. For instance, the insecure expression $(v_B := (!\ u_A))$ is not bisimilar to itself if $A \not\sqsubseteq^F B$.

We now present a weakened version of non-disclosure for networks, that is defined over thread configurations.

Definition 3.3 (Non-disclosure for Networks (on thread configurations))**.** *A pool of threads P satisfies the Non-disclosure for Networks property with respect to an allowed-policy mapping W, a reference labeling Σ, a thread labeling Υ and a typing environment Γ, if it satisfies $\langle P,T_1\rangle \approx^{\Sigma,\Upsilon}_{\Gamma,l} \langle P,T_2\rangle$ for all security levels l and position trackers T_1,T_2 such that $\mathrm{dom}(P)=\mathrm{dom}(T_1)=\mathrm{dom}(T_2)$ and $T_1 =^{\Sigma,\Upsilon}_{\mho,l} T_2$. We then write $P\in\mathcal{NDN}_2(W,\Sigma,\Upsilon,\Gamma)$.*

Properties. Definition 3.3 is strictly weaker, in the sense that it considers more programs as secure, than the old thread pool-based definition of [4]. The idea is that if we only consider reasonable locations for the thread at the point that the leak is performed, then we are accepting more programs. In the following result, the sets $\mathcal{NDN}_1(W,\Sigma,\Upsilon,\Gamma)$ and $\mathcal{NDN}_2(W,\Sigma,\Upsilon,\Gamma)$ are the sets of secure pools of threads according to Definition 3.3 and the one in [4], respectively.

Proposition 3.4. $\mathcal{NDN}_1(W, \Sigma, \Upsilon, \Gamma) \subset \mathcal{NDN}_2(W, \Sigma, \Upsilon, \Gamma)$.

Example 3 illustrates which programs are now deemed secure by the new property, that remembers the possible locations at each point of the bisimulation game.

The new weakened definition of Non-disclosure for Networks is compositional with respect to set union of pools of threads, up to disjoint naming of threads.

Example. We are considering a simplistic memory model where all of the network's memory is accessible at all times by every process in the network. With this assumption we avoid *migration leaks* that derive from synchronization behaviors [4]. In our setting, we avoid these issues, but migration leaks can be encoded nonetheless. The idea is that now a program can reveal information about the position of a thread in a network by performing tests on the flow policy that is allowed by that site:

$$\begin{array}{r}\text{if } (!\ x_H) \text{ then } (\text{thread}_l \text{ (allowed } F \text{ then } (y_L := 1) \text{ else } (y_L := 2)) \text{ at } d_1) \\ \text{else } (\text{thread}_l \text{ (allowed } F \text{ then } (y_L := 1) \text{ else } (y_L := 2)) \text{ at } d_2)\end{array} \quad (4)$$

In this example, the new thread will be created at (migrate) to domains d_1 or d_2 depending on the tested high value; then, if these domains have different allowed flow policies, different low-assignments are performed, thus revealing high level information. Therefore, the program is insecure with respect to Non-disclosure for Networks.

3.2. Type and Effect System

We now present a type and effect system [27] that accepts programs that satisfy Non-disclosure for Networks, as defined in Subsubsection 3.1. It is similar to the one in [3], for a language that is extended to include the allowed conditions, and while restricting information leaks to occur within the boundaries of the flow declarations, as in [4]. An additional parameter appears in this setting, where the reference labeling is made explicit.

The typing judgments used in Figure 4 have the form

$$\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$$

meaning that the expression M is typable with type τ and security effect s in the typing context $\Gamma : \boldsymbol{Var} \to \boldsymbol{Typ}$, which assigns types to variables. The turnstile has three parameters: (1) the reference labeling Σ; (2) the flow policy *declared by the context* F, represents the one that is valid in the evaluation context in which the expression M is typed, and contributes to the meaning of operations and relations on security levels. (3) he security level j represents the confidentiality level associated to the thread that the expression M is part of, i.e. the confidentiality level of the location of that thread in the network.

The security effect s is composed of three security levels that are referred to by $s.r$, $s.w$ and $s.t$, and can be understood as follows: $s.r$ is the *reading effect*, an upper-bound on the security levels of the references that are read by M; $s.w$ is the *writing effect*, a lower bound on the references that are written by M; $s.t$ is the *termination effect*, an upper bound on the level of the references

$$[\textsc{Nil}]\ \Gamma \vdash^{\Sigma}_{j,F} () : s, \mathsf{unit} \qquad [\textsc{BT}]\ \Gamma \vdash^{\Sigma}_{j,F} tt : s, \mathsf{bool} \qquad [\textsc{BF}]\ \Gamma \vdash^{\Sigma}_{j,F} ff : s, \mathsf{bool}$$

$$[\textsc{Loc}]\ \Gamma \vdash^{\Sigma}_{j,F} a : s, \Sigma_2(a)\ \mathrm{ref}_{\Sigma_1(a)} \qquad [\textsc{Var}]\ \Gamma, x : \tau \vdash^{\Sigma}_{j,F} x : s, \tau$$

$$[\textsc{Abs}]\ \frac{\Gamma, x : \tau \vdash^{\Sigma}_{j,F} M : s, \sigma}{\Gamma \vdash^{\Sigma}_{j',F'} (\lambda x.M) : s', \tau \xrightarrow[j,F]{s} \sigma} \qquad [\textsc{Rec}]\ \frac{\Gamma, x : \tau \vdash^{\Sigma}_{j,F} W : s, \tau}{\Gamma \vdash^{\Sigma}_{j,F} (\varrho x.W) : s, \tau}$$

$$[\textsc{Flow}]\ \frac{\boldsymbol{\Gamma \vdash^{\Sigma}_{j,F \curlywedge F'} N : s, \tau}}{\boldsymbol{\Gamma \vdash^{\Sigma}_{j,F} (\mathbf{flow}\ F'\ \mathbf{in}\ N) : s, \tau}}$$

$$[\textsc{Allow}]\ \frac{\begin{array}{c}\boldsymbol{\Gamma \vdash^{\Sigma}_{j,F} N_t : s_t, \tau} \\ \boldsymbol{\Gamma \vdash^{\Sigma}_{j,F} N_f : s_f, \tau}\end{array} \quad \boldsymbol{j \sqsubseteq^{F} s_t.w, s_f.w}}{\boldsymbol{\Gamma \vdash^{\Sigma}_{j,F} (\mathbf{allowed}\ F'\ \mathbf{then}\ N_t\ \mathbf{else}\ N_f) : s_t \sqcup s_f \sqcup \langle \bot, \top, j \rangle, \tau}}$$

$$[\textsc{Ref}]\ \frac{\Gamma \vdash^{\Sigma}_{j,F} M : s, \theta \quad s.r, s.t \sqsubseteq^{F} l}{\Gamma \vdash^{\Sigma}_{j,F} (\mathrm{ref}_{l,\theta}\ M) : s \sqcup \langle \bot, l, \bot \rangle, \theta\ \mathrm{ref}_l} \qquad [\textsc{Der}]\ \frac{\Gamma \vdash^{\Sigma}_{j,F} M : s, \theta\ \mathrm{ref}_l}{\Gamma \vdash^{\Sigma}_{j,F} (!\ M) : s \sqcup \langle l, \top, \bot \rangle, \theta}$$

$$[\textsc{Ass}]\ \frac{\Gamma \vdash^{\Sigma}_{j,F} M : s, \theta\ \mathrm{ref}_l \quad \Gamma \vdash^{\Sigma}_{j,F} N : s', \theta \qquad \begin{array}{r} s.t \sqsubseteq^{F} s'.w \\ s.r, s'.r, s.t, s'.t \sqsubseteq^{F} l \end{array}}{\Gamma \vdash^{\Sigma}_{j,F} (M := N) : s \sqcup s' \sqcup \langle \bot, l, \bot \rangle, \mathsf{unit}}$$

$$[\textsc{Cond}]\ \frac{\Gamma \vdash^{\Sigma}_{j,F} M : s, \mathsf{bool} \quad \begin{array}{c}\Gamma \vdash^{\Sigma}_{j,F} N_t : s_t, \tau \\ \Gamma \vdash^{\Sigma}_{j,F} N_f : s_f, \tau\end{array} \quad s.r, s.t \sqsubseteq^{F} s_t.w, s_f.w}{\Gamma \vdash^{\Sigma}_{j,F} (\mathrm{if}\ M\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f) : s \sqcup s_t \sqcup s_f \sqcup \langle \bot, \top, s.r \rangle, \tau}$$

$$[\textsc{Seq}]\ \frac{\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau \quad \Gamma \vdash^{\Sigma}_{j,F} N : s', \sigma \quad s.t \sqsubseteq^{F} s'.w}{\Gamma \vdash^{\Sigma}_{j,F} (M; N) : s \sqcup s', \sigma}$$

$$[\textsc{App}]\ \frac{\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau \xrightarrow[j,F]{s'} \sigma \quad \Gamma \vdash^{\Sigma}_{j,F} N : s'', \tau \qquad \begin{array}{r} s.t \sqsubseteq^{F} s''.w \\ s.r, s''.r, s.t, s''.t \sqsubseteq^{F} s'.w \end{array}}{\Gamma \vdash^{\Sigma}_{j,F} (M\ N) : s \sqcup s' \sqcup s'' \sqcup \langle \bot, \top, s.r \sqcup s''.r \rangle, \sigma}$$

$$[\textsc{Mig}]\ \frac{\boldsymbol{\Gamma \vdash^{\Sigma}_{l,\mho} M : s, \mathsf{unit}}}{\boldsymbol{\Gamma \vdash^{\Sigma}_{j,F} (\mathbf{thread}_l\ M\ \mathbf{at}\ d') : \langle \bot, l \sqcup s.w, \bot \rangle, \mathsf{unit}}}$$

Figure 4: Type and Effect System for checking Non-disclosure for Networks

on which the termination of expression M might depend. According to these intuitions, in the type system the reading and termination levels are composed in a covariant way, whereas the writing level is contravariant.

Types have the following syntax (t is a type variable):

$$\tau, \sigma, \theta \ \in\ \boldsymbol{Typ}\ ::=\ t \mid \mathsf{unit} \mid \mathsf{bool} \mid \theta\ \mathrm{ref}_l \mid \tau \xrightarrow[j,F]{s} \sigma$$

Typable expressions that reduce to $()$ have type unit, and those that reduce to booleans have type bool. Typable expressions that reduce to a reference which points to values of type θ and has security level l have the reference type $\theta\ \mathrm{ref}_l$. The security level l is used to determine the effects of expressions that handle

references. Expressions that reduce to a function that takes a parameter of type τ, that returns an expression of type σ, and with a *latent effect* s [27] have the function type $\tau \xrightarrow[j,F]{s} \sigma$. The latent effect is the security effect of the body of the function, while the latent flow policies are those that are assumed to hold when the function is applied to an argument, and the latent security level j of the thread containing the expression that appears in the the type of expressions that reduce to functions.

We use a lattice on security effects, that is obtained from the point-wise composition of three lattices of the security levels. More precisely:

$$s \sqsubseteq^F s' \overset{\text{def}}{\Leftrightarrow} s.r \sqsubseteq^F s'.r \;\&\; s'.w \sqsubseteq^F s.w \;\&\; s.t \sqsubseteq^F s'.t$$
$$s \sqcup s' \overset{\text{def}}{\Leftrightarrow} \langle s.r \sqcup s'.r, s.w \sqcap s'.w, s.t \sqcup s'.t \rangle$$
$$\top = \langle \top, \bot, \top \rangle \qquad \bot = \langle \bot, \top, \bot \rangle$$

Our type and effect system applies restrictions to programs in order to enforce compliance of all information flows to the flow relation that is parameterized with the current flow policy. This is achieved by conditions of the kind "$\sqsubseteq^F$" in the premises of the typing rules, and by the update of the security effects in the conclusions. Apart from the parameterization of the flow relation with the current flow policy, these are fairly standard in information flow type system and enforce syntactic rules of the kind "no low writes should depend on high reads", both with respect to the values that are read, and to termination behaviors that might be derived. Notice that the FLOW rule types the body of the flow declaration under a more permissive flow policy. We refer the reader to [4] for explanations on all of these conditions. There are, however, extra conditions that are introduced in order to deal with new forms of migration leaks that appear in our distributed setting (such as Example 4), and that deserve further attention: the security level j that is associated to each thread, and represents the confidentiality level of the position of the thread in the network is used to update the termination effect in the allowed-condition rule, for the choice of the branch can determine the termination behavior of the condition. The security level of the thread is also constrained not to be higher in confidentiality than the "low writes" in rule ALLOW. The extra condition regarding the termination effect (such as in rules ASS, COND and APP) are in fact, they are implicit in the previous type system as well. Here they must appear explicitly since it is no longer true that $s.t \sqsubseteq^F s.r$ for any F, due to the update of the termination effect with the level j in rule ALLOW.

We are now in position to define the compatibility predicate that applies to this particular information flow analysis:

Definition 3.5 ((Σ, Γ)-Compatibility)**.** *A memory S is said to be* (Σ, Γ)-compatible *if, for every reference $a \in \mathrm{dom}(S)$, its value $S(a)$ satisfies $\Gamma \vdash_\Sigma S(a) : \Sigma_2(a)$.*

3.2.1. Soundness

The main result of this section, soundness, states that the type system only accepts expressions that are secure in the sense of Definition 3.3. In the remainder of this section we sketch the main definitions and results that can be used

to reconstruct a direct proof of this result. A similar proof is given in detail for a similar language (without the allowed condition or remote thread creation) in [4].

Subject Reduction. In order to establish the soundness of the type system of Figure 4 we need a Subject Reduction result, stating that the type of a thread is preserved by reduction. When a thread performs a computation step, some of its effects may be performed by reading, updating or creating a reference, and some may be discarded when a branch in a conditional expression is taken. Then the effects of an expression "weaken" along the computations. To prove it we follow the usual steps [41].

Proposition 3.6 (Subject Reduction)**.** *Given a reference and thread labeling* Σ, Υ, *consider a thread* M^m *for which there exist* Γ, F, s *and* τ *such that* $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$ *and suppose that* $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F']{d} \langle \{M'^m\} \cup P, T', S'\rangle$, *for a memory* S *that is* (Σ, Γ)*-compatible. Then, there is an effect* s' *such that* $s' \sqsubseteq s$ *and* $\Gamma \vdash^{\Sigma}_{j,F} M' : s', \tau$, *and* S' *is also* (Σ, Γ)*-compatible. Furthermore, if* $P = \{N^n\}$, *for some expression* N *and thread name* n, *then there exists* s'' *such that* $s.w \sqsubseteq s''.w$ *such that* $\Gamma \vdash^{\Sigma}_{\Upsilon(k),\mho} N : s'', \mathsf{unit}$.

Properties of the Semantics. One can prove that the semantics preserves the conditions for well-formedness, and that a configuration with a single expression has at most one transition, up to the choice of new names.

The following result states that, if the evaluation of a thread M^m differs in the context of two distinct states while not creating two distinct reference names or thread names, this is because either M^m is performing a dereferencing operation, which yields different results depending on the memory, or because M^m is testing the allowed policy.

Lemma 3.7 (Splitting Computations)**.**
If we have $W \vdash \langle \{M^m\}, T_1, S_1\rangle \xrightarrow[F]{d} \langle P_1', T_1', S_1'\rangle$ *and* $W \vdash \langle \{M^m\}, T_2, S_2\rangle \xrightarrow[F']{d}$ $\langle P_2', T_2', S_2'\rangle$ *with* ${P_1}' \neq {P_2}'$, *then* $P_1' = \{{M_1}'^m\}$, $P_2' = \{{M_2}'^m\}$ *and either:*

- $\exists \mathrm{E}, a$ *such that* $F = \lceil \mathrm{E} \rceil = F'$, $M = \mathrm{E}[(!\ a)]$, *and* $M_1' = \mathrm{E}[S_1(a)]$, $M_2' = \mathrm{E}[S_2(a)]$ *with* $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$ *and* $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$, *or*
- $\exists \mathrm{E}, \bar{F}$ *such that* $F = \lceil \mathrm{E} \rceil = \bar{F}$, $M = \mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$, *and* $T_1(m) \neq T_2(m)$ *with* $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$ *and* $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$.

Proof. Note that the only rules that depend on the state are those for the reduction of $\mathrm{E}[(!\ a)]$ and of $\mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$. By case analysis on the transition $W \vdash \langle \{M^m\}, T_1, S_1\rangle \xrightarrow[F]{d} \langle P_1', T_1', S_1'\rangle$. □

High Expressions. We can identify a class of threads that have the property of never performing any change in the "low" part of the memory. These are classified as being "high" according to their behavior:

Definition 3.8 (Operationally High Threads)**.** *A set of threads $\mathcal{H}^{\Sigma,\Upsilon}$ is a set of* operationally (Σ, Υ, F, l)-high threads *if the following holds for all $M^m \in \mathcal{H}^{\Sigma,\Upsilon}$, for all states $\langle T, S\rangle$, and for all flow policy mappings W:*

$$W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F']{d} \langle P', T', S'\rangle \text{ implies } \langle T, S\rangle =^{\Sigma,\Upsilon}_{F,l} \langle T', S'\rangle \text{ and } P' \subseteq \mathcal{H}^{\Sigma,\Upsilon}$$

The largest set of operationally (Σ, Υ, F, l)-high threads is denoted by $\mathcal{H}^{\Sigma,\Upsilon}_{F,l}$. We then say that a thread M^m is operationally (Σ, Υ, F, l)-high, if $M^m \in \mathcal{H}^{\Sigma_1,\Upsilon}_{F,l}$.

For any Σ, Υ, F and l, the set of threads with values as expressions is a set of operationally (Σ, Υ, F, l)-high threads. Furthermore, the union of a family of sets of operationally (Σ, Υ, F, l)-high threads is a set of operationally (Σ, Υ, F, l)-high threads. Consequently, $\mathcal{H}^{\Sigma,\Upsilon}_{F,l}$ exists.

Notice that if $F' \subseteq F$, then any operationally (Σ, Υ, F, l)-high thread is also operationally $(\Sigma, \Upsilon, F', l)$-high.

Some expressions can be easily classified as "high" by the type system, which only considers their syntax. These cannot perform changes to the "low" memory simply because their code does not contain any instruction that could perform them. Since the writing effect is intended to be a lower bound to the level of the references that the expression can create or assign to, expressions with a high writing effect can be said to be *syntactically high*:

Definition 3.9 (Syntactically High Expressions)**.** *An expression M is syntactically $(\Sigma, \Gamma, j, F, l)$-high if there exists s, τ such that $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$ with $s.w \not\sqsubseteq^F l$. The expression M is a syntactically $(\Sigma, \Gamma, j, F, l)$-high function if there exists j', F', s, τ, σ such that $\Gamma \vdash^{\Sigma}_{j',F'} M : \tau \xrightarrow[j,F]{s} \sigma$ with $s.w \not\sqsubseteq^F l$.*

Syntactically high expressions have an operationally high behavior.

Lemma 3.10 (High Expressions)**.** *If M is a syntactically (Σ,Γ, j, F, l)-high expression, and $\Upsilon(m) = j$, then M^m is an operationally (Σ, Υ, F, l)-high thread.*

Proof. We show that the following is a set of operationally (Σ, Υ, F, l)-high threads:

$$\{M^m \mid \exists j \ .\ M \text{ is syntactically } (\Sigma,\Gamma, j, F, l)\text{-high}\}$$

□

Behavior of "Low"-Terminating Expressions. We first build a symmetric binary relation between typable expressions whose terminating behaviors do not depend on high references, more precisely, between those that are typable with a low termination effect. The binary relation should be such that if the evaluation of two related expressions, in the context of two low-equal stores should split (see Lemma 3.7), then the resulting expressions are still in the relation. This relation, called $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ is inductively defined for a security level *low* as follows:

Definition 3.11 ($\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$)**.** *We have that $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ if $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1, \tau$ and $\Gamma \vdash^{\Sigma}_{j,F} M_2 : s_2, \tau$ for some s_1, s_2 and τ with $s_1.t \sqsubseteq^F$ low and $s_2.t \sqsubseteq^F$ low and one of the following holds:*

1. M_1 *and* M_2 *are both values, or*
2. $M_1 = M_2$*, or*
3. $M_1 = (\bar{M}_1; \bar{N})$ *and* $M_2 = (\bar{M}_2; \bar{N})$ *where* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, or*
4. $M_1 = (\mathrm{ref}_{l,\theta} \ \bar{M}_1)$ *and* $M_2 = (\mathrm{ref}_{l,\theta} \ \bar{M}_2)$ *where* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, and* $l \not\sqsubseteq^F$ *low, or*
5. $M_1 = (!\ \bar{M}_1)$ *and* $M_2 = (!\ \bar{M}_2)$ *where* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, or*
6. $M_1 = (\bar{M}_1 := \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 := \bar{N}_2)$ *with* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, and* $\bar{N}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{N}_2$*, and* $\bar{M}_1, \bar{M}_2$ *both have type* $\theta \ \mathrm{ref}_l$ *for some* θ *and* l *such that* $l \not\sqsubseteq^F$ *low, or*
7. $M_1 = (\mathrm{flow}\ F' \ \mathrm{in}\ \bar{M}_1)$ *and* $M_2 = (\mathrm{flow}\ F' \ \mathrm{in}\ \bar{M}_2)$ *with* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F \curlywedge F',low} \ \bar{M}_2$*.*

The next proposition states that $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ is a kind of "strong bisimulation" with respect to the transition relation $\xrightarrow[F']{d}$.

Proposition 3.12 (Strong Bisimulation for Low-Termination)**.** *Suppose that* $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ *and, for a given allowed flow policy mapping* W*, also* $W \vdash \langle \{{M_1}^m\}, T_1, S_1\rangle \xrightarrow[F']{d} \langle P_1', T_1', S_1'\rangle$*, with* $\Upsilon(m) = j$ *and* $\langle T_1, S_1\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2, S_2\rangle$*. Then* $P_1' = \{{M_1'}^m\} \cup P$*, and if* $a \in \mathrm{dom}(S_1' - S_1)$ *implies that* a *is fresh for* S_2*, then there exist* M_2'*,* T_2' *and* S_2' *such that* $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m\}, T_2', S_2'\rangle$ *with* $M_1' \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2'$ *and* $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$*. Furthermore, if* $P = \{N^n\}$ *for some expression* N *and thread name* $n \notin \mathrm{dom}(T_2)$*, then there exist* M_2'*,* T_2' *and* S_2' *such that* $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m, N^n\}, T_2', S_2'\rangle$ *with* $M_1' \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2'$ *and* $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$*.*

Proof. By case analysis on the clause by which $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$, and by induction on the definition of $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$. □

Behavior of Typable Low Expressions. We now define a larger symmetric binary relation on typable expressions. Similarly to the previous one, it should be possible to relate the results of the computations of two related expressions in the context of two low-equal memories. The binary relation $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$ on expressions is defined inductively as follows:

Definition 3.13 ($\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$)**.** *We have that* $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ *if* $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1, \tau$ *and* $\Gamma \vdash^{\Sigma}_{j,F} M_2 : s_2, \tau$ *for some* Γ*,* s_1*,* s_2 *and* τ *and one of the following holds:*

1. ${M_1}^m, {M_2}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$*, or*
2. $M_1 = M_2$*, or*
3. $M_1 = (\mathrm{if}\ \bar{M}_1 \ \mathrm{then}\ \bar{N}_t \ \mathrm{else}\ \bar{N}_f)$ *and also* $M_2 = (\mathrm{if}\ \bar{M}_2 \ \mathrm{then}\ \bar{N}_t \ \mathrm{else}\ \bar{N}_f)$ *with* $\bar{M}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, and* ${\bar{N}_t}^m, {\bar{M}_f}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$*, or*
4. $M_1 = (\bar{M}_1 \ \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 \ \bar{N}_2)$ *with* $\bar{M}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, and* $\bar{N}_1^m, \bar{N}_2^m \in \mathcal{H}_{F,low}$*, and* $\bar{M}_1, \bar{M}_2$ *are syntactically* (F, low, j)*-high functions, or*
5. $M_1 = (\bar{M}_1 \ \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 \ \bar{N}_2)$ *with* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, and* $\bar{N}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{N}_2$*, and* $\bar{M}_1, \bar{M}_2$ *are syntactically* (F, low, j)*-high functions, or*
6. $M_1 = (\bar{M}_1; \bar{N})$ *and* $M_2 = (\bar{M}_2; \bar{N})$ *with* $\bar{M}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$*, and* $\bar{N}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$*, or*

7. $M_1 = (\bar{M}_1; \bar{N})$ *and* $M_2 = (\bar{M}_2; \bar{N})$ *with* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$, *or*
8. $M_1 = (\mathrm{ref}_{l,\theta} \ \bar{M}_1)$ *and* $M_2 = (\mathrm{ref}_{l,\theta} \ \bar{M}_2)$ *with* $\bar{M}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$, *and* $l \not\sqsubseteq^F low$, *or*
9. $M_1 = (! \ \bar{M}_1)$ *and* $M_2 = (! \ \bar{M}_2)$ *with* $\bar{M}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$, *or*
10. $M_1 = (\bar{M}_1 := \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 := \bar{N}_2)$ *with* $\bar{M}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$, *and* $\bar{N}_1^m, \bar{N}_2^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, *and* $\bar{M}_1, \bar{M}_2$ *both have type* $\theta \ \mathrm{ref}_{l,n_k}$ *for some* θ *and* l *such that* $l \not\sqsubseteq^F low$, *or*
11. $M_1 = (\bar{M}_1 := \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 := \bar{N}_2)$ *with* $\bar{M}_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M}_2$, $\bar{N}_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \bar{N}_2$, *and* $\bar{M}_1, \bar{M}_2$ *both have type* $\theta \ \mathrm{ref}_{l,n_k}$ *for some* θ *and* l *such that* $l \not\sqsubseteq^F low$, *or*
12. $M_1 = (\mathrm{flow} \ F' \ \mathrm{in} \ \bar{M}_1)$ *and* $M_2 = (\mathrm{flow} \ F' \ \mathrm{in} \ \bar{M}_2)$ *with* $\bar{M}_1 \ \mathcal{R}^{j}_{F \curlywedge F',low} \ \bar{M}_2$.

The relation $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$ is a kind of "strong bisimulation", with respect to the transition relation $\xrightarrow[F']{d}$:

Proposition 3.14 (Strong Bisimulation for Typable Low Threads)**.** *Suppose that* $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ *and* $M_1 \notin \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$ *and for a given allowed flow policy mapping* W, *also* $W \vdash^{\Sigma,\Upsilon} \langle \{{M_1}^m\}, T_1, S_1\rangle \xrightarrow[F']{d} \langle P_1', T_1', S_1'\rangle$, *with* $\Upsilon(m) = j$ *and* $\langle T_1, S_1\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2, S_2\rangle$. *Then* $P_1' = \{{M_1'}^m\} \cup P$, *and if* $a \in \mathrm{dom}(S_1' - S_1)$ *implies that* a *is fresh for* S_2, *then there exist* M_2', T_2' *and* S_2' *such that* $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m\}, T_2', S_2'\rangle$ *with* $M_1' \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} M_2'$ *and* $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$. *Furthermore, if* $P = \{N^n\}$ *for some expression* N *and thread name* $n \notin \mathrm{dom}(T_2)$, *then there exist* M_2', T_2' *and* S_2' *such that* $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m, N^n\}, T_2', S_2'\rangle$ *with* $M_1' \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2'$ *and* $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$.

Proof. By case analysis on the clause by which $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$, and by induction on the definition of $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$. □

Behavior of Sets of Typable Threads. To conclude the proof of the Soundness Theorem, it remains to exhibit an appropriate bisimulation on thread configurations.

Definition 3.15 ($\mathcal{R}^{\Upsilon}_{low}$)**.** *The relation* $\mathcal{R}^{\Upsilon}_{low}$ *is inductively defined as follows:*

$$a) \ \frac{M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}}{\{M^m\} \ \mathcal{R}^{\Upsilon}_{low} \ \emptyset} \qquad b) \ \frac{M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}}{\emptyset \ \mathcal{R}^{\Upsilon}_{low} \ \{M^m\}}$$

$$c) \ \frac{M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),\mho,low} \ M_2}{\{{M_1}^m\} \ \mathcal{R}^{\Upsilon}_{low} \ \{{M_2}^m\}} \qquad d) \ \frac{P_1 \ \mathcal{R}^{\Upsilon}_{low} \ P_2 \quad Q_1 \ \mathcal{R}^{\Upsilon}_{low} \ Q_2}{P_1 \cup Q_1 \ \mathcal{R}^{\Upsilon}_{low} \ P_2 \cup Q_2}$$

We will now use Strong Bisimulation for Typable Low Threads (Proposition 3.14) to prove the following:

Proposition 3.16. *The relation*

$$\mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} = \{(\langle P_1, T_1\rangle, \langle P_2, T_2\rangle) \mid P_1 \ \mathcal{R}^{\Upsilon}_{low} \ P_2 \text{ and } T_1 =^{\Sigma,\Upsilon}_{\mho,l} T_2\}$$

is a $(W, \Sigma, \Upsilon, \Gamma, l)$*-bisimulation acording to Definition 3.2.*

We can now prove the main result regarding Non-disclosure for Networks:

Theorem 3.17 (Soundness of Typing Non-disclosure for Networks.)**.** *Consider a pool of threads P, an allowed-policy mapping W, a reference labeling Σ, a thread labeling Υ and a typing environment Γ. If for all $M^m \in P$ there exist s, and τ such that $\Gamma \vdash^{\Sigma}_{\Upsilon(m),\mho} M : s, \tau$, then P satisfies the Non-disclosure for Networks policy, i.e. $P \in \mathcal{NDN}_2(W, \Sigma, \Upsilon, \Gamma)$.*

Proof. For all $M^m \in P$ and for all choices of security levels *low*, by assumption and by Clause 2' of Definition 3.13, we have that $M \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; M$. By Rule c) of Definition 3.15 we then have $\{M^m\} \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \{M^m\}$. Therefore, by Rule d) we have that $P \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; P$, from which we conclude that for all position trackers T_1, T_2 such that $\mathrm{dom}(P) = \mathrm{dom}(T_1) = \mathrm{dom}(T_2)$ and $T_1 =^{\Sigma,\Upsilon}_{\mho,l} T_2$ we have $\langle P, T_1\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle P, T_2\rangle$. By Proposition 3.16 we conclude that $\langle P, T_1\rangle \approx^{\Sigma,\Upsilon}_{\Gamma,low} \langle P, T_2\rangle$. □

Our soundness result for non-disclosure is compositional, in the sense that it is enough to verify the typability of each thread separately in order to ensure non-disclosure for the whole network.

4. Controlling Declassification

We now define flow policy confinement with respect to a setting with distributed allowed flow policies, and justify the chosen formalization. The property is similar to the one in [2], but is now formulated in terms of thread configurations [6]. We study its enforcement y means by means of three migration control mechanisms. We start by studying a type system for statically ensuring that global computations always comply to the locally valid allowed flow policy. This type system is inherently restrictive, as the domains where each part of the code will actually compute cannot in general be known statically (Subsection 4.2). We then present a more precise type system to be used at runtime by the semantics of the language for checking migrating threads against the allowed flow policy of the destination domain (Subsection 4.3). Finally, we propose a yet more precise type and effect system that computes information about the declassification behaviors of programs. This information will be used more efficiently at runtime by the semantics of the language in order to control migration of programs (Subsection 4.4).

4.1. Flow Policy Confinement for Networks

The property of Flow Policy Confinement states that the declassifications that are declared by a program at each computation step comply to the allowed policy of the domain where the step is performed. In a distributed setting with concurrent mobile code, programs might need to comply simultaneously to different allowed flow policies that change dynamically. We deal with this difficulty by placing individual restrictions on each step that might be performed

by a part of the program, taking into account the possible location where it might take place.

Memory compatibility. Similarly to Subsection 3.1, memories are assumed to be compatible to the given security setting and typing environment, requiring typability of their contents according to the relevant enforcement mechanism. This predicate will be defined for each security analysis that is performed over the next three subsections (see Definitions 4.4, 4.8 and 4.10).

Definition on thread configurations. Similarly to [5] we define the property co-inductively, on thread configurations. The location of each thread determines which allowed flow policy it should obey at that point, and is used to place a restriction on the flow policies that decorate the transitions.

Definition 4.1 ((W, Σ, Γ)-Confined Thread Configurations)**.** *Given an allowed-policy mapping W, a reference labeling Σ, and a typing environment Γ, a set $\mathcal{CTC}$ of thread configurations is a set of* (W, Σ, Γ)-confined thread configurations *if it satisfies, for all P, T, and (W, Σ, Γ)-compatible stores S:*

$$\langle P, T\rangle \in \mathcal{CTC} \text{ and } W \vdash \langle P, T, S\rangle \xrightarrow[F]{d} \langle P', T', S'\rangle \text{ implies}$$
$$W(d) \preccurlyeq F \text{ and } \langle P', T'\rangle \in \mathcal{CTC}$$

Furthermore, S' is still (W, Σ, Γ)-compatible. The largest set of (W, Σ, Γ)-confined thread configurations is denoted $\mathcal{CTC}_W^{\Sigma,\Gamma}$.

For any W, Σ and Γ, the set of thread configurations where threads are values is a set of (W, Σ, Γ)-confined thread configurations. Furthermore, the union of a family of (W, Σ, Γ)-confined thread configurations is a (W, Σ, Γ)-confined thread configurations. Consequently, $\mathcal{CTC}_W^{\Sigma,\Gamma}$ exists.

Definition 4.2 (Flow Policy Confinement)**.** *A pool of threads P satisfies Flow Policy Confinement with respect to an allowed-policy mapping W, a reference labeling Σ and a typing environment Γ, if all thread configurations satisfy $\langle P, T\rangle \in \mathcal{CTC}_W^{\Sigma,\Gamma}$. We then write $P \in \mathcal{FPC}_2(W, \Sigma, \Gamma)$.*

Notice that the property is parameterized by a particular mapping W from domains to allowed flow policies. This means that security is defined relative to W. An absolute notion of security holds when W is universally quantified.

It should be clear that Flow Policy Confinement speaks strictly about what *flow declarations* a thread can do *while* it is at a specific domain. In particular, it does not restrict threads from migrating to more permissive domains in order to perform a declassification. It does not deal with information flows. So for instance it offers no assurance that information leaks that are encoded at each point of the program do obey the declared flow policies for that point. For example, program

$$(\text{thread}_l\ (\text{flow } F \text{ in } (b := (!\ a))) \text{ at } d) \tag{5}$$

always satisfies flow policy confinement when $F = \mathfrak{V}$, regardless of the levels of references a and b.

$$[\textsc{Nil}]\ W;\Gamma \vdash^{\Sigma}_{A} () : \mathsf{unit} \qquad [\textsc{BT}]\ W;\Gamma \vdash^{\Sigma}_{A} tt : \mathsf{bool} \qquad [\textsc{BF}]\ W;\Gamma \vdash^{\Sigma}_{A} ff : \mathsf{bool}$$

$$[\textsc{Loc}]\ W;\Gamma \vdash^{\Sigma}_{A} a : \Sigma_2(a)\ \mathrm{ref} \qquad [\textsc{Var}]\ W;\Gamma, x:\tau \vdash^{\Sigma}_{A} x : \tau$$

$$[\textsc{Abs}]\ \frac{W;\Gamma, x:\tau \vdash^{\Sigma}_{A} M : \sigma}{W;\Gamma \vdash^{\Sigma}_{A'} (\lambda x.M) : \tau \underset{A}{\rightarrow} \sigma} \qquad [\textsc{Rec}]\ \frac{W;\Gamma, x:\tau \vdash^{\Sigma}_{A} W : \tau}{W;\Gamma \vdash^{\Sigma}_{A} (\varrho x.W) : \tau}$$

$$[\textsc{Flow}]\ \frac{\boldsymbol{W;\Gamma \vdash^{\Sigma}_{A} N : \tau} \quad \boldsymbol{A \preccurlyeq F}}{\boldsymbol{W;\Gamma \vdash^{\Sigma}_{A} (\mathbf{flow}\ F\ \mathbf{in}\ N) : \tau}}$$

$$[\textsc{Allow}]\ \frac{\begin{array}{c}\boldsymbol{W;\Gamma \vdash^{\Sigma}_{A \curlywedge F} N_t : \tau} \\ \boldsymbol{W;\Gamma \vdash^{\Sigma}_{A} N_f : \tau}\end{array}}{\boldsymbol{W;\Gamma \vdash^{\Sigma}_{A} (\mathbf{allowed}\ F\ \mathbf{then}\ N_t\ \mathbf{else}\ N_f) : \tau}}$$

$$[\textsc{Ref}]\ \frac{W;\Gamma \vdash^{\Sigma}_{A} M : \theta}{W;\Gamma \vdash^{\Sigma}_{A} (\mathrm{ref}_{l,\theta}\ M) : \theta\ \mathrm{ref}} \qquad [\textsc{Der}]\ \frac{W;\Gamma \vdash^{\Sigma}_{A} M : \theta\ \mathrm{ref}}{W;\Gamma \vdash^{\Sigma}_{A} (!\ M) : \theta}$$

$$[\textsc{Ass}]\ \frac{W;\Gamma \vdash^{\Sigma}_{A} M : \theta\ \mathrm{ref} \quad W;\Gamma \vdash^{\Sigma}_{A} N : \theta}{W;\Gamma \vdash^{\Sigma}_{A} (M := N) : \mathsf{unit}} \qquad [\textsc{Seq}]\ \frac{W;\Gamma \vdash^{\Sigma}_{A} M : \tau \quad W;\Gamma \vdash^{\Sigma}_{A} N : \sigma}{W;\Gamma \vdash^{\Sigma}_{A} (M;N) : \sigma}$$

$$[\textsc{App}]\ \frac{W;\Gamma \vdash^{\Sigma}_{A} M : \tau \underset{A}{\rightarrow} \sigma \quad W;\Gamma \vdash^{\Sigma}_{A} N : \tau}{W;\Gamma \vdash^{\Sigma}_{A} (M\ N) : \sigma}$$

$$[\textsc{Cond}]\ \frac{W;\Gamma \vdash^{\Sigma}_{A} M : \mathsf{bool} \qquad \begin{array}{c} W;\Gamma \vdash^{\Sigma}_{A} N_t : \tau \\ W;\Gamma \vdash^{\Sigma}_{A} N_f : \tau \end{array}}{W;\Gamma \vdash^{\Sigma}_{A} (\mathrm{if}\ M\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f) : \tau}$$

$$[\textsc{Mig}]\ \frac{\boldsymbol{W;\Gamma \vdash^{\Sigma}_{A'} M : \mathsf{unit}} \quad \boldsymbol{W(d) \preccurlyeq A'}}{\boldsymbol{W;\Gamma \vdash^{\Sigma}_{A} (\mathbf{thread}_l\ M\ \mathbf{at}\ d) : \mathsf{unit}}}$$

Figure 5: Type and effect system for checking Confinement

Properties. Flow Policy Confinement, defined over thread configurations, is equivalent to when defined over located threads. In the following result, the sets $\mathcal{FPC}_1(W,\Sigma,\Gamma)$ and $\mathcal{FPC}_2(W,\Sigma,\Gamma)$ are the sets of secure pools of threads according to Definition 3.3 and the one in [5], respectively.

Proposition 4.3. $\mathcal{FPC}_1(W,\Sigma,\Gamma) = \mathcal{FPC}_2(W,\Sigma,\Gamma)$.

4.2. Static Type and Effect System

We have seen that in a setting where code can migrate between domains with different allowed security policies, the computation domain might change *during* computation, along with the allowed flow policy that the program must comply to. This can happen in particular within the branch of an allowed condition:

$$(\text{allowed } F \text{ then } (\text{thread}_l\ (\text{flow } F \text{ in } M_1) \text{ at } d) \text{ else } M_2) \tag{6}$$

In this program, the flow declaration of the policy F is executed only if F has been tested as being allowed by the domain where the program was started. It might then seem that the flow declaration is "protected" by an appropriate allowed construct. However, by the time the flow declaration is performed, the thread is already located at another domain, where that flow policy might not be allowed. It is clear that a static enforcement of a confinement property requires tracking the possible locations where threads might be executing at each point.

Figure 5 presents a new type and effect system [27] for statically enforcing confinement over a migrating program. The type system guarantees that when operations are executed by a thread within the scope of a flow declaration, the declared flow complies to the allowed flow policy of the current domain. The typing judgments have the form

$$W; \Gamma \vdash^{\Sigma}_{A} M : \tau$$

meaning that the expression M is typable with type τ in the typing context $\Gamma : \boldsymbol{Var} \to \boldsymbol{Typ}$, which assigns types to variables, in a context where W is the mapping of domain names to allowed flow policies. The turnstile has two parameters: (1) the reference labeling Σ; (2) the flow policy *allowed by the context* A, which includes all flow policies that have been positively tested by the program as being allowed at the computation domain where the expression M is running.

Types have the following syntax (t is a type variable):

$$\tau, \sigma, \theta \ \in \ \boldsymbol{Typ} \ ::= \ t \mid \mathsf{unit} \mid \mathsf{bool} \mid \theta \ \mathsf{ref} \mid \tau \underset{A}{\longrightarrow} \sigma$$

The syntax is similar to the one used in Subsubsection 3.2, but is simpler: The security level of references does not appear in the reference types θ ref, while the type function types $\tau \underset{A}{\longrightarrow} \sigma$ includes only the latent allowed flow policy, the one that is assumed to hold when the function is applied to an argument.

Our type and effect system applies restrictions to programs in order to enforce confinement of all flow declarations of a policy F to be performed only once F has been tested to be positively allowed by the domain's allowed flow policy. This is achieved by means of the tested allowed flow policy A that parameterizes the typing judgments, and by the condition $A \preccurlyeq F'$ in the FLOW rule. Notice that the ALLOW rule types the "allowed" branch of the condition under an extended allowed flow policy. Flow declarations can only be performed if the declared flow policy is allowed by the flow policy that is tested by the context (FLOW), while allowed conditions relax the typing of the first branch by extending the flow policy that is tested by the context with the policy that guards the condition (ALLOW). The difference between the two type systems is mainly in rule MIG, which initializes the policy that is allowed by the computation domain with that of the destination domain $W(d)$ that is specified by the $(\mathrm{thread}_l\ M \text{ at } d)$ construct.

Note that if an expression is typable with respect to an allowed flow policy A, then it is also so for any more permissive allowed policy A'. In particular, due to the ABS rule, the process of typing an expression is not deterministic. For instance, the expression $(\lambda x.())$ can be given any type of the form $\tau \underset{A}{\longrightarrow} \mathsf{unit}$.

We refer to the enforcement mechanism that consists of statically type checking all threads in a network according to the type and effect system of Figure 5, with respect to the allowed flow policies of each thread's initial domain, using the semantics represented in Figure 3, as *Enforcement mechanism I.*

To illustrate the restrictions that are imposed by the enforcement mechanism, we may consider program

$$(\text{allowed } F \text{ then } (\text{flow } F_{H \prec L} \text{ in } \textit{plan\_A}) \text{ else } \textit{plan\_B}) \tag{7}$$

where *plan_A* and *plan_B* have no declassifications. The program is typable if $W(d) \curlywedge F \preccurlyeq F_{H \prec L}$. If $F \preccurlyeq F_{H \prec L}$, then the program is always secure. Otherwise, the program is W-secure if $W(d) \preccurlyeq F_{H \prec L}$.

We are now in position to define the compatibility predicate that applies to enforcement mechanism I.

Definition 4.4 ((W, Σ, Γ)-Compatibility)**.** *A memory S is said to be* (W, Σ, Γ)-compatible *if, for every reference $a \in \mathrm{dom}(S)$, its value $S(a)$ satisfies the typing condition $W; \Gamma \vdash^{\Sigma}_{\mho,} S(a) : \Sigma_2(a)$.*

4.2.1. Soundness

Subject reduction. In order to establish the soundness of the type system of Figure 5 we need a Subject Reduction result, stating that types that are given to expressions are preserved by computation. To prove it we follow the usual steps [41].

We check that the type of a thread and the compatibility of memories is preserved by reduction.

Proposition 4.5 (Subject Reduction)**.** *Given a reference and thread labeling Σ, Υ, consider a thread M^m for which there exist Γ, A and τ such that $W; \Gamma \vdash^{\Sigma}_{A} M : \tau$ and suppose that $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{M'^m\} \cup P, T', S'\rangle$, for a memory S that is (W, Σ, Γ)-compatible. Then, $W; \Gamma \vdash^{\Sigma}_{A \curlywedge W(T(m))} M' : \tau$, and S' is also (W, Σ, Γ)-compatible. Furthermore, if $P = \{N^n\}$, for some expression N and thread name n, then $W; \Gamma \vdash^{\Sigma}_{W(T'(n))} N : \mathsf{unit}$.*

Proof. We follow the usual steps [41], where the main proof is a case analysis on the transition $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{M'^m\} \cup P, T', S'\rangle$. □

Soundness. Enforcement mechanism I guarantees security of networks with respect to confinement, as is formalized by the following result.

Theorem 4.6 (Soundness of Enforcment Mechanism I)**.** *Consider a fixed allowed-policy mapping W, a given reference labeling Σ and typing environment Γ, and a thread configuration $\langle P, T\rangle$ such that for all $M^m \in P$ there exists τ such that $W; \Gamma \vdash^{\Sigma}_{W(T(m))} M : \tau$. Then $\langle P, T\rangle$ is a (W, Σ, Γ)-confined thread configuration.*

Proof. Consider the following set:

$$C = \{\langle P, T\rangle \mid \forall M^m \in P, \exists \tau \,.\, W; \Gamma \vdash^{\Sigma}_{W(T(m))} M : \tau\}$$

We show that C is a set of (W, Σ, Γ)-confined thread configurations. If for a given (W, Σ, Γ)-compatible store S we have that there exist P', T', S' such that $W \vdash \langle P, T, S\rangle \xrightarrow[F]{d} \langle P', T', S'\rangle$, then, there is a thread M^m such that $P = \{M^m\} \cup \bar{P}$ and $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{M^m\} \cup \bar{P}', T', S'\rangle$, with $P' = \{M^m\} \cup \bar{P}' \cup \bar{P}$ and $T(m) = d$. By induction on the inference of $W; \Gamma \vdash^{\Sigma}_{W(T(m))} M : \tau$, we prove that $W(d) \preccurlyeq F$, and $W; \Gamma \vdash^{\Sigma}_{W(T'(m))} M' : \tau$. Furthermore, if $\bar{P} = N^n$ for some expression N and thread name N, then $W; \Gamma \vdash^{\Sigma}_{W(T'(n))} N : \tau$.

Since typability of the threads that result from the transition step, as well as the (W, Σ, Γ)-compatibility of the stores, is guaranteed by Subject Reduction (Proposition 4.5), we prove only the conditions regarding the compliance of the declared flow policies to the current domain's allowed flow policy. Assuming that $W; \Gamma \vdash^{\Sigma}_{W(d)} M : \tau$, and considering the last rule in the corresponding typing proof:

Flow. Here $M = (\text{flow } \bar{F} \text{ in } \bar{M})$, and we have $W; \Gamma \vdash^{\Sigma}_{W(d)} \bar{M} : \tau$, with $W(d) \preccurlyeq \bar{F}$. There are two cases to consider:

- $\bar{M}$ **can compute.** Then $W \vdash \langle \{\bar{M}\}, T, S\rangle \xrightarrow[\bar{F}']{d} \langle \{\bar{M}'\} \cup \bar{P}', T', S'\rangle$, with $F = \bar{F} \curlywedge \bar{F}'$. By induction hypothesis, then $W(d) \preccurlyeq \bar{F}'$. Since $W(d) \preccurlyeq \bar{F}$, then $W(d) \preccurlyeq F$.
- $\bar{M} \in$ ***Val***. Then we have $F = \mho$, so $W(T(m)) \preccurlyeq F$ holds vacuously.

Allow. Here $M = (\text{allowed } \bar{A} \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$, and we have and $F = \mho$. Therefore $W(T(m)) \preccurlyeq F$ holds vacuously.

Mig. In this case $M = (\text{thread}_l\ \bar{M} \text{ at } \bar{d})$, with $W; \Gamma \vdash^{\Sigma} \bar{M} : \mathsf{unit}$. Then we have $F = \mho$, so $W(T(m)) \preccurlyeq F$ holds vacuously.

The cases for Ref, Der, Ass, Seq, App and Cond are similar to Flow, since for the cases where the sub-expressions are not all values, these sub-expressions require typing assumptions that have the same A parameter as the concluding judgment, and for the cases where the expression is ready to reduce, the flow policy that decorates the transition is $\mho$. The cases for Rec and Mig is similar to Allow, since the constructs are not evaluation contexts, and therefore the transition is decorated with the top flow policy $F = \mho$.

□

Precision. Given the purely static nature of this migration control analysis, some secure programs are bound to be rejected. There are different ways to increase the precision of a type system, which are all intrinsically limited to what can conservatively be predicted before runtime. For example, for the program

$$(\text{if } (!\, a) \text{ then } (\text{thread}_l\ (\text{flow } F \text{ in } M) \text{ at } d_1) \text{ else } (\text{thread}_l\ (\text{flow } F \text{ in } M) \text{ at } d_2)) \quad (8)$$

it is in general not possible to predict which branch will be executed (or, in practice, to which domain the thread will migrate), for it depends on the contents of the memory. It will then be rejected if $W(d_2) \not\preccurlyeq F$ or $W(d_1) \not\preccurlyeq F$.

$$\vdots$$

$$[\textsc{Mig}]\ \frac{\boldsymbol{\Gamma} \vdash^{\boldsymbol{\Sigma}}_{\boldsymbol{\Omega}} \boldsymbol{M} : \mathsf{unit}}{\boldsymbol{\Gamma} \vdash^{\boldsymbol{\Sigma}}_{\boldsymbol{A}} (\mathbf{thread}_{\boldsymbol{l}}\ \boldsymbol{M}\ \mathbf{at}\ \boldsymbol{d}) : \mathsf{unit}}$$

Figure 6: Relaxed Type and Effect System for Checking Confinement (Fragment)

4.3. Runtime Type Checking

In this subsection we study a hybrid mechanism for enforcing confinement, that makes use of a relaxation of the type system of Figure 5 at runtime. Migration is now controlled by means of a runtime check for typability of migrating threads with respect to the allowed flow policy of the destination domain. The condition represents the standard theoretical requirement of checking incoming code before allowing it to execute in a given machine.

The relaxation is achieved by replacing rule Mig by the one in Figure 6: The new type system no longer imposes *future* migrating threads to conform to the policy of their destination domain, but only to the most permissive allowed flow policy Ω. The rationale is that it only worries about confinement of the non-migrating parts of the program. This is sufficient, as all threads that are to be spawned by the program will be re-checked at migration time.

The following modification to the migration rule of the semantics of Figure 3 introduces the runtime check that controls migration (n fresh in T). The idea is that a thread can only migrate to a domain if it respects its allowed flow policy:

$$\frac{\Gamma \vdash^{\Sigma}_{W(d)} N : \mathsf{unit}}{W \vdash^{\Sigma,\Upsilon} \langle \{\mathrm{E}[(\mathrm{thread}_l\ N\ \mathrm{at}\ d)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle \{\mathrm{E}[()]^m, N^n\}, [n := d]T, S\rangle} \tag{9}$$

The new remote thread creation rule (our migration primitive), now depends on typability of the migrating thread. The typing environment Γ (which is constant) is now an implicit parameter of the operational semantics. If only closed threads are considered, then also migrating threads are closed. The allowed flow policy of the destination site now determines whether or not a migration instruction may be consummated, or otherwise block execution. E.g., the configuration

$$\langle \{\mathrm{E}[(\mathrm{thread}_l\ (\mathrm{flow}\ F\ \mathrm{in}\ M)\ \mathrm{at}\ d)]^m\}, T, S\rangle \tag{10}$$

can only proceed if $W(d)$ allows for F; otherwise it gets stuck.

Notice that, thanks to postponing the migration control to runtime, the type system no longer needs to be parameterized with information about the allowed flow policies of all domains in the network, which in practice could be impossible. The only relevant ones are those of the destination domain of migrating threads.

We refer to the enforcement mechanism that consists of statically type checking all threads in a network according to the type and effect system of Figure 5 modified using the new Mig rule represented in Figure 6, with respect to the allowed flow policies of each thread's initial domain, using the semantics of Figure 3 modified according to Figure 6, as *Enforcement mechanism II*.

Notice that enforcement mechanism II restricts, on one hand, which programs are accepted to run, but also trims their possible executions, with respect to a given allowed-policy mapping W. Program 8 illustrates this mechanism, as it is typable according to the relaxed type system (with respect to any W), but will block at the choice of the second branch if $W(d_2) \not\preccurlyeq F$.

4.3.1. Soundness
Subject Reduction.

Proposition 4.7 (Subject Reduction)**.** *Given a reference and thread labeling Σ, Υ, consider a thread M^m for which there exist Γ, A and τ such that $\Gamma \vdash^{\Sigma}_{A} M : \tau$ and suppose that $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{M'^m\} \cup P, T', S'\rangle$, for a memory S that is (Σ, Γ)-compatible. Then, $\Gamma \vdash^{\Sigma}_{A \curlywedge W(T(m))} M' : \tau$, and S' is also (Σ, Γ)-compatible. Furthermore, if $P = \{N^n\}$, for some expression N and thread name n, then $\Gamma \vdash^{\Sigma}_{W(T'(n))} N : \mathsf{unit}$.*

Proof. We follow the usual steps [41], where the main proof is a case analysis on the transition $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F']{d} \langle \{M'^m\} \cup P, T', S'\rangle$. □

Soundness. We are now in position to define the compatibility predicate that applies to enforcement mechanism II.

Definition 4.8 ((W, Σ, Γ)-Compatibility)**.** *A memory S is said to be* (W, Σ, Γ)-compatible *if, for every reference $a \in \mathrm{dom}(S)$, its value $S(a)$ satisfies the typing condition $\Gamma \vdash^{\Sigma}_{\mho} S(a) : \Sigma_2(a)$.*

Enforcement mechanism II guarantees security of networks with respect to confinement, as is formalized by the following result.

Theorem 4.9 (Soundness of Enforcement Mechanism II)**.** *Consider a fixed allowed-policy mapping W, a given reference labeling Σ and typing environment Γ, and a thread configuration $\langle P, T\rangle$ such that for all $M^m \in P$ there exists τ such that $\Gamma \vdash^{\Sigma}_{W(T(m))} M : \tau$. Then $\langle P, T\rangle$ is (W, Σ, Γ)-confined.*

Proof. The same as for Theorem 4.6, where it is shown that the set

$$C = \{\langle P, T\rangle \mid \forall M^m \in P, \exists \tau \,.\, \Gamma \vdash^{\Sigma}_{W(T(m))} M : \tau\}$$

is a set of (W, Σ, Γ)-confined thread configurations. □

4.3.2. Safety, precision and efficiency

The proposed mechanism does not offer a safety result, guaranteeing that programs never "get stuck". Indeed, the side condition of the thread creation rule introduces the possibility for the execution of a thread to block, since no alternative is given. This can happen in Example 6 (in page 25), if the flow policy F is not permitted by the allowed policy of the domain of the branch that is actually executed, then the migration will not occur, and execution will not proceed. In order to have safety, we could design the thread creation instruction

as including an alternative branch for execution in case the side condition fails. Nevertheless, Example 6 might have better been written

$$(\text{thread}_l\ (\text{allowed}\ F\ \text{then}\ (\text{flow}\ F\ \text{in}\ M_1)\ \text{else}\ M_2)\ \text{at}\ d)$$

in effect using the allowed condition for encoding such alternative behaviors. Indeed, the programmer can increase the chances that the program will be allowed to migrate to other sites by protecting flow declarations with the new allowed condition construct. Our type system does take that effort into account, by "omitting" the tested flow policy when typing the "allowed" branch. As a result, programs containing flow declarations that are too permissive according to a certain domain might still be authorized to execute in it, as long as they occur in the "not allowed" branch of our new construct which will not be chosen.

Returning to Example 8 (in page 28), thanks to the relaxed MIG rule, this program is now *always* accepted statically by the type system. Depending on the result of the test, the migration might also be allowed to occur if a safe branch is chosen. This means that enforcement mechanism II accepts more secure programs.

It is worth noting that the change in the semantics of the thread creation introduces new information flow leaks, as the event of blockage of a computation can reveal information about the control flow that led to it. These information leaks can be treated in a similar manner as termination leaks, as in [2], for blockage can be seen as a form of non-termination. Such leaks are in fact rejected by the type and effect system of Figure 4 (though the soundness result that is presented in this paper does not cover it). It is clear, however, that not all programs that can block encode information leaks. To see this it suffices to consider a security mapping that assigns the same security level to all references in the program.

A drawback with this enforcement mechanism lies in the computation weight of the runtime type checks. This is particularly acute for an expressive language such as the one we are considering. Indeed, recognizing typability of ML expressions has exponential (worst case) complexity [28].

4.4. Static Informative Typing for Runtime Effect Checking

We have seen that bringing the type-based migration control of programs to runtime allows to increase the precision of the confinement analysis. This is, however, at the cost of performance. It is possible to separate the program analysis as to what are the declassification operations that are performed by migrating threads, from the safety problem of determining whether those declassification operations should be allowed at a given domain. To achieve this, we now present an *informative* type system [7] that statically calculates a summary of all the declassification operations that might be performed by a program, in the form of a *declassification effect*. Furthermore, this type system produces a version of the program that is annotated with the relevant information for deciding, at runtime, whether its migrating threads can be considered safe by the destination domain. The aim is to bring the overhead of the runtime check to static time.

$$[\textsc{Nil}_\textsc{I}]\ \Gamma \vdash^\Sigma () \hookrightarrow () : \mho, \mathsf{unit} \quad [\textsc{Bt}_\textsc{I}]\ \Gamma \vdash^\Sigma tt \hookrightarrow tt : \mho, \mathsf{bool} \quad [\textsc{Bf}_\textsc{I}]\ \Gamma \vdash^\Sigma ff \hookrightarrow ff : \mho, \mathsf{bool}$$

$$[\textsc{Loc}_\textsc{I}]\ \Gamma \vdash^\Sigma a \hookrightarrow a : \mho, \Sigma_2(a)\ \mathrm{ref} \qquad [\textsc{Var}_\textsc{I}]\ \Gamma, x : \tau \vdash^\Sigma x \hookrightarrow x : \mho, \tau$$

$$[\textsc{Abs}_\textsc{I}]\ \frac{\Gamma, x : \tau \vdash^\Sigma M \hookrightarrow \hat{M} : s, \sigma}{\Gamma \vdash^\Sigma (\lambda x.M) \hookrightarrow (\lambda x.\hat{M}) : \mho, \tau \xrightarrow{s} \sigma} \qquad [\textsc{Rec}_\textsc{I}]\ \frac{\Gamma, x : \tau \vdash^\Sigma X \hookrightarrow \hat{X} : s, \tau}{\Gamma \vdash^\Sigma (\varrho x.X) \hookrightarrow (\varrho x.\hat{X}) : s, \tau}$$

$$[\textsc{Ref}_\textsc{I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \theta' \quad \theta \preccurlyeq \theta'}{\Gamma \vdash^\Sigma (\mathrm{ref}_\theta\ M) \hookrightarrow (\mathrm{ref}_\theta\ \hat{M}) : s, \theta\ \mathrm{ref}} \qquad [\textsc{Der}_\textsc{I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \theta\ \mathrm{ref}}{\Gamma \vdash^\Sigma (!\ M) \hookrightarrow (!\ \hat{M}) : s, \theta}$$

$$[\textsc{Ass}_\textsc{I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \theta\ \mathrm{ref} \quad \Gamma \vdash^\Sigma N \hookrightarrow \hat{N} : s', \theta' \quad \theta \preccurlyeq \theta'}{\Gamma \vdash^\Sigma (M := N) \hookrightarrow (\hat{M} := \hat{N}) : s \curlywedge s', \mathsf{unit}}$$

$$[\textsc{Cond}_\textsc{I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \mathsf{bool} \qquad \begin{array}{c}\Gamma \vdash^\Sigma N_t \hookrightarrow \hat{N}_t : s_t, \tau_t \\ \Gamma \vdash^\Sigma N_f \hookrightarrow \hat{N}_f : s_f, \tau_f\end{array} \qquad \tau_t \approx \tau_f}{\Gamma \vdash^\Sigma (\mathrm{if}\ M\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f) \hookrightarrow (\mathrm{if}\ \hat{M}\ \mathrm{then}\ \hat{N}_t\ \mathrm{else}\ \hat{N}_f) : s \curlywedge s_t \curlywedge s_f, \tau_t \curlywedge \tau_f}$$

$$[\textsc{Seq}_\textsc{I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \tau \quad \Gamma \vdash^\Sigma N \hookrightarrow \hat{N} : s', \sigma}{\Gamma \vdash^\Sigma (M; N) \hookrightarrow : s \curlywedge s', \sigma}$$

$$[\textsc{App}_\textsc{I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \tau \xrightarrow{s'} \sigma \quad \Gamma \vdash^\Sigma N \hookrightarrow \hat{N} : s'', \tau'' \quad \tau \preccurlyeq \tau''}{\Gamma \vdash^\Sigma (M\ N) \hookrightarrow (\hat{M}\ \hat{N}) : s \curlywedge s' \curlywedge s'', \sigma}$$

$$[\mathbf{Flow_I}]\ \frac{\Gamma \vdash^\Sigma N \hookrightarrow \hat{N} : s, \tau}{\Gamma \vdash^\Sigma (\mathrm{flow}\ F\ \mathrm{in}\ N) \hookrightarrow (\mathrm{flow}\ F\ \mathrm{in}\ \hat{N}) : s \curlywedge F, \tau}$$

$$[\mathbf{Allow_I}]\ \frac{\begin{array}{c}\Gamma \vdash^\Sigma N_t \hookrightarrow \hat{N}_t : s_t, \tau_t \\ \Gamma \vdash^\Sigma N_f \hookrightarrow \hat{N}_f : s_f, \tau_f\end{array} \qquad \tau_t \approx \tau_f}{\begin{array}{l}\Gamma \vdash^\Sigma (\mathrm{allowed}\ F\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f) \hookrightarrow (\mathrm{allowed}\ F\ \mathrm{then}\ \hat{N}_t\ \mathrm{else}\ \hat{N}_f) : \\ \qquad\qquad s_t \backsim F \curlywedge s_f, \tau_t \curlywedge \tau_f\end{array}}$$

$$[\mathbf{Mig_I}]\ \frac{\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \mathsf{unit}}{\Gamma \vdash^\Sigma (\mathrm{thread}_l\ M\ \mathrm{at}\ d) \hookrightarrow (\mathrm{thread}^s_l\ \hat{M}\ \mathrm{at}\ d) : \mho, \mathsf{unit}}$$

Figure 7: Informative Type and Effect System for obtaining the Declassification Effect

The typing judgments of the type system in Figure 7 have the form:

$$\Gamma \vdash^\Sigma M \hookrightarrow \hat{M} : s, \tau$$

Comparing with the typing judgments of Subsection 4.3, while the flow policy allowed by the context parameter is omitted from the turnstile '$\vdash$', the security effect s represents a flow policy which corresponds to the *declassification effect*: a lower bound to the flow policies that are declared in the typed expression. The second expression $\hat{M}$ is the result of annotating M. We thus consider an annotated version of the language of Subsection 2.2, where the syntax of values and expressions (the sets are denoted $\boldsymbol{Val'}$ and $\boldsymbol{Exp'}$, respectively) differs only in the remote thread creation construct.

Types have the following syntax (t is a type variable):

$$\tau, \sigma, \theta \ \in \ \textbf{\textit{Typ}} \ ::= \ t \mid \mathsf{unit} \mid \mathsf{bool} \mid \theta \ \mathsf{ref} \mid \tau \xrightarrow{s} \sigma$$

The syntax of annotated expressions differs only in the thread creation construct, that has an additional policy F as parameter, written $(\mathrm{thread}_l^F \ M \text{ at } d)$. The syntax of types is the same as the one used in Subsections 4.2 and 4.3.

It is possible to relax the type system by matching types that have the same structure, even if they differ in flow policies pertaining to them. We achieve this by overloading $\preccurlyeq$ to relate types where certain latent effects in the first are at least as permissive as the corresponding ones in the second. The more general relation $\approx$ matches types where certain latent effects differ: Finally, we define an operation $\curlywedge$ between two types τ and τ' such that $\tau \approx \tau'$:

$$\tau \preccurlyeq \tau' \text{ iff } \tau = \tau', \text{ or } \tau = \theta \xrightarrow{F} \sigma \text{ and } \tau' = \theta \xrightarrow{F'} \sigma' \text{ with } F \preccurlyeq F' \text{ and } \sigma \preccurlyeq \sigma'$$

$$\tau \approx \tau' \text{ iff } \tau = \tau', \text{ or } \tau = \theta \xrightarrow{F} \sigma \text{ and } \tau' = \theta \xrightarrow{F'} \sigma' \text{ with } \sigma \approx \sigma'$$

$$\tau \curlywedge \tau' = \tau, \text{ if } \tau = \tau', \text{ or } \theta \xrightarrow{F \curlywedge F'} \sigma \curlywedge \sigma', \text{ if } \tau = \theta \xrightarrow{F} \sigma \text{ and } \tau' = \theta \xrightarrow{F'} \sigma'$$

The $\preccurlyeq$ relation is used in rules $\textsc{Ref}_\textsc{I}$, $\textsc{Ass}_\textsc{I}$ and $\textsc{App}_\textsc{I}$, in practice enabling to associate to references and variables (by reference creation, assignment and application) expressions with types that contain stricter policies than required by the declared types. The relation $\approx$ is used in rules $\textsc{Cond}_\textsc{I}$ and$\textsc{Allow}_\textsc{I}$ in order to accept that two branches of the same test construct can differ regarding some of their policies. Then, the type of the test construct is constructed from both using $\curlywedge$, thus reflecting the flow policies in both branches.

The declassification effect is constructed by aggregating (using the meet operation) all relevant flow policies that are declared within the program. The effect is updated in rule $\textsc{Flow}_\textsc{I}$, each time a flow declaration is performed, and "grows" as the declassification effects of sub-expressions are met in order to form that of the parent command. However, when a part of the program is "protected" by an allowed condition, some of the information in the declassification effect can be discarded. This happens in rule $\textsc{Allow}_\textsc{I}$, where the declassification effect of the first branch is not used entirely: the part that will be tested during execution by the allowed-condition is omitted. In rule $\textsc{Mig}_\textsc{I}$, the declassification effect of migrating threads is also not recorded in the effect of the parent program, as they will be executed (and tested) elsewhere. That information is however used to annotate the migration instruction.

As an example, the thread creation of program 7, still assuming that *plan_A* and *plan_B* have no declassifications, would be annotated with the declassification effect $F_{H \prec L} \smile F$. In particular, the effect would be $\mho$ if $F \preccurlyeq F_{H \prec L}$.

One can show that the type system is deterministic, in the sense that it assigns to a non-annotated expression a single annotated version of it, a single declassification effect, and a single type.

4.4.1. Modified operational semantics, revisited.

By executing annotated programs, the type check that conditions the migration instruction can be replaced by a simple declassification effect inspection.The new migration rule is similar to the one in Subsection 4.3, but now makes use of the declassification effect (n fresh in T):

$$\frac{W(d) \preccurlyeq s}{W \vdash^{\Sigma,\Upsilon} \langle\{\mathrm{E}[(\mathrm{thread}_l^s\ N \text{ at } d)]^m\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{T(m)} \langle\{\mathrm{E}[()]^m, N^n\}, [n := d]T, S\rangle} \quad (11)$$

In the remaining rules of the operational semantics the annotations are ignored. Note that the values contained in memories are also assumed to use annotated syntax.

We refer to the mechanism that consists of statically annotating all threads in a network according to the type and effect system of Figure 7, assuming that each thread's declassification effect is allowed by its initial domain, using the semantics of Figure 3 modified according to Rule (11), as *Enforcement mechanism III*.

We are now in position to define the compatibility predicate that applies to enforcement mechanism III.

Definition 4.10 ((W, Σ, Γ)-Compatibility)**.** *A memory S is said to be* (W, Σ, Γ)-compatible *if, for every reference $a \in \mathrm{dom}(S)$, its value $S(a)$ results from annotating some other value V according to $\Gamma \vdash^{\Sigma}_{\mho,} V \hookrightarrow S(a) : \Sigma_2(a)$.*

4.4.2. Soundness

Subject Reduction. The following proposition ensures that the annotation processing is preserved by the annotated semantics. This is formulated by stating that after reduction, programs are still well annotated. Since the type system integrates both the annotating process and the calculation of the declassification effect, the formulation of this result is slightly non-standard. More precisely, the following result states that if a program is the result of an annotation process, a certain declassification effect and type, then after one computation step it is still the result of annotating a program, and is given a not-more permissive declassification effect and type.

Proposition 4.11 (Subject Reduction, or Preservation of Annotations)**.** *Given an allowed-policy mapping W, a reference labeling Σ and a typing environment Γ, consider a thread M^m for which there exist N, s and τ such that $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$ and suppose $W \vdash^{\Sigma,\Upsilon} \langle\{N^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{N'^m\} \cup P, T', S'\rangle$, for a memory S that is (Σ, Γ)-compatible. Then there exist M', s', τ' such that $s \curlywedge W(T(m)) \preccurlyeq s'$, and $\tau \preccurlyeq \tau'$, and $\Gamma \vdash^{\Sigma} M' \hookrightarrow N' : s', \tau'$, and S' is also (Σ, Γ)-compatible. Furthermore, if $P = \{N''^n\}$ for some expression N'' and thread name n, then there exist M'', s'' such that $W(T'(n)) \preccurlyeq s''$ and $\Gamma \vdash^{\Sigma} M'' \hookrightarrow N'' : s'', \mathsf{unit}$.*

Proof. We follow the usual steps [41], where the main proof is a case analysis on the transition $W \vdash \langle\{M^m\}, T, S\rangle \xrightarrow[F']{d} \langle\{M'^m\} \cup P, T', S'\rangle$. □

Soundness. We will now see that the declassification effect can be used for enforcing confinement.

Theorem 4.12 (Soundness of Enforcement Mechanism III)**.** *Consider a fixed allowed-policy mapping W, a given reference labeling Σ and typing environment*

Γ, and a thread configuration $\langle P, T\rangle$ such that for all $M^m \in P$ there exist $\hat{M}$, s and τ such that $\Gamma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau$ and $W(T(m)) \preccurlyeq s$. Then $\langle \hat{P}, T\rangle$, formed by annotating the threads in $\langle P, T\rangle$, is (W, Σ, Γ)-confined.

Proof. Consider the following set:

$$C = \{\langle P, T\rangle \mid \forall \hat{M^m} \in P,\ \exists M, s, \tau\ .\ \Gamma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau \text{ and } W(T(m)) \preccurlyeq s\}$$

We show that C is a set of (W, Σ, Γ)-confined thread configurations. As in the proof of Theorem 4.6, if for a given (W, Σ, Γ)-compatible store S we have that there exist P', T', S' such that $W \vdash \langle P, T, S\rangle \xrightarrow[F]{d} \langle P', T', S'\rangle$, then, there is a thread M^m such that $P = \{M^m\} \cup \bar{P}$ and $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{M^m\} \cup \bar{P}', T', S'\rangle$, with $P' = \{M^m\} \cup \bar{P}' \cup \bar{P}$ and $T(m) = d$. By induction on the inference of $\Gamma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau$, we prove that $W(d) \preccurlyeq F$ and there exists M', s' and τ' such that $\Gamma \vdash^{\Sigma} M' \hookrightarrow \hat{M'} : s', \tau'$ with $W(T'(m)) \preccurlyeq s'$. Furthermore, if $\bar{P} = N^n$ for some expression N and thread name N, then there exists M', s', τ' such that $\Gamma \vdash^{\Sigma} M' \hookrightarrow \hat{M'} : s', \tau'$ with $W(T'(m)) \preccurlyeq s$, and there exists N, s'' such that $\Gamma \vdash^{\Sigma} N \hookrightarrow \hat{N} : s'', \mathsf{unit}$ with $W(T'(n)) \preccurlyeq s''$.

Notice that since we necessarily have $T'(m) = T(m) = d$, then typability of the threads that result from the transition step is guaranteed by Subject Reduction (Proposition 4.11). Also, by the same result, we have compliance of the new declassification effect s' to the domain's allowed flow policy, for: If $s \curlywedge W(T(m)) \preccurlyeq s'$, and since $W(d) = W(T(m)) \preccurlyeq s$, then $W(T(m)) \preccurlyeq s'$. Furthermore, if $\bar{P} = N^n$, then $W(T'(n)) \preccurlyeq s''$.

It remains to prove the conditions regarding the compliance of the declared flow policies to the current domain's allowed flow policy. Assuming that $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$ and $W(T(m)) \preccurlyeq s$ and considering the last rule in the corresponding typing proof:

Flow$_\mathbf{I}$. Here $M = (\text{flow } \bar{F} \text{ in } \bar{M})$, $N = (\text{flow } \bar{F} \text{ in } \bar{N})$, and $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \tau$, with $s = \bar{s} \curlywedge \bar{F}$. Then, we have that $W(d) \preccurlyeq \bar{s}$. There are two cases to consider:

- $\bar{N}$ **can compute.** Then $W \vdash^{\Sigma,\Upsilon} \langle \{\bar{N}^m\}, T, S\rangle \xrightarrow[\bar{F}']{d} \langle \{\bar{N'}^m\} \cup P, T', S'\rangle$, with $F = \bar{F} \curlywedge \bar{F}'$. By induction hypothesis, then $W(T(m)) \preccurlyeq \bar{F}'$. Since $W(T(m)) = W(d) \preccurlyeq \bar{F}$, then $W(T(m)) \preccurlyeq F$.
- $\bar{N} \in \boldsymbol{Val}$. Then we have $F = \mho$, so $W(T(m)) \preccurlyeq F$ holds vacuously.

Allow$_\mathbf{I}$. Here $N = (\text{allowed } \bar{A} \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$, and we have $F = \mho$. Therefore $W(T(m)) \preccurlyeq F$ holds vacuously.

Mig$_\mathbf{I}$. In this case $M = (\text{thread}_l\ \bar{M} \text{ at } \bar{d})$ and $N = (\text{thread}_l^{\bar{s}}\ \bar{N} \text{ at } \bar{d})$, with $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \mathsf{unit}$. Then we have $F = \mho$, so $W(T(m)) \preccurlyeq F$ holds vacuously. Since N can reduce, then $W(\bar{d}) = W(T'(n)) \preccurlyeq \bar{s}$.

The cases for REF$_\mathrm{I}$, DER$_\mathrm{I}$, ASS$_\mathrm{I}$, SEQ$_\mathrm{I}$, APP$_\mathrm{I}$ and COND$_\mathrm{I}$ are similar to FLOW$_\mathrm{I}$, since for the cases where the sub-expressions are not all values, these

sub-expressions are typed with a declassification effect s' that is at least as restrictive as that s' of the concluding judgment, i.e. $s \preccurlyeq s'$, and so since $W(d) \preccurlyeq s'$ then the induction hypothesis can be applied. Furthermore, for the cases where the sub-expression is ready to reduce, the flow policy that decorates the transition is the same as F. When the sub-expressions are all values (more precisely, those that are to be evaluated in the evaluation context provided by each of these constructs), then $F = \mho$. The case for $\textsc{Rec}_{\textsc{I}}$ is similar to $\textsc{Allow}_{\textsc{I}}$, since the constructs are not evaluation contexts, and therefore the transition is decorated with the top flow policy $F = \mho$.

□

4.4.3. Precision and efficiency

The relaxed type system of Subsection 4.3 for checking confinement, and its informative counterpart of Figure 7, are strongly related. The following result states that typability according to latter type system is at least as precise as the former.

Proposition 4.13. *Consider a given a typing environment* Γ *and reference labeling* Σ*. If there exist* A*,* τ *such that* $W; \Gamma \vdash^{\Sigma}_{A,} M : \tau$*, then there exist* $\hat{M}$*,* τ' *and* s *such that* $\Gamma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau'$ *and* $A \preccurlyeq s$ *with* $\tau \preccurlyeq \tau'$*.*

Proof. By induction on the inference of $\Gamma \vdash^{\Sigma}_{A} M : \tau$, and by case analysis on the last rule used in this typing proof. We show the cases for the non-standard constructs.

Mig. Here $M = (\text{thread}_l\ \bar{M}\ \text{at}\ d)$ and we have that $W; \Gamma \vdash^{\Sigma}_{W(d)} \bar{M} : \bar{\tau}$, with $\tau = \bar{\tau} = \mathsf{unit}$. By induction hypothesis, there exist $\bar{N}$, $\bar{s}$, $\bar{\tau}'$ such that $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \bar{\tau}'$ and $W(d) \preccurlyeq \bar{s}$ with $\bar{\tau} \preccurlyeq \bar{\tau}'$. We conclude using rule $\textsc{Mig}_{\textsc{I}}$, $N = (\text{thread}^{\bar{s}}_{l}\ \bar{N}\ \text{at}\ d)$, $s = \mho$ and $\tau' = \mathsf{unit}$.

Flow. Here $M = (\text{flow}\ \bar{F}\ \text{in}\ \bar{M})$ and $W; \Gamma \vdash^{\Sigma}_{A} \bar{M} : \bar{\tau}$, with $A \preccurlyeq \bar{F}$ and $\tau = \bar{\tau}$. By induction hypothesis, there exist $\bar{N}$, $\bar{s}$, $\bar{\tau}'$ such that $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \bar{\tau}'$ and $A \preccurlyeq \bar{s}$ with $\bar{\tau} \preccurlyeq \bar{\tau}'$. Therefore $A \preccurlyeq \bar{s} \curlywedge \bar{F}$ and we conclude using rule $\textsc{Flow}_{\textsc{I}}$, $N = (\text{flow}\ \bar{F}\ \text{in}\ \bar{N})$, $s = \bar{s} \curlywedge \bar{F}$ and $\tau' = \bar{\tau}'$.

Allow. Here $M = (\text{allowed}\ \bar{F}\ \text{then}\ \bar{M}_t\ \text{else}\ \bar{M}_f)$ and we have $W; \Gamma \vdash^{\Sigma}_{A \curlywedge \bar{F}} \bar{M}_t : \bar{\tau}$ and $W; \Gamma \vdash^{\Sigma}_{A} \bar{M}_f : \bar{\tau}$ where $\bar{\tau} = \tau$. By induction hypothesis, there exist $\bar{N}_t$, $\bar{N}_f$, $\bar{s_t}$, $\bar{s_f}$ and $\bar{\tau}'$ such that $\Gamma \vdash^{\Sigma} \bar{M}_t \hookrightarrow \bar{N}_t : \bar{s_t}, \bar{\tau}'$ and $\Gamma \vdash^{\Sigma} \bar{M}_f \hookrightarrow \bar{N}_f : \bar{s_f}, \bar{\tau}'$ and $A \curlywedge \bar{F} \preccurlyeq \bar{s_t}'$, $A \preccurlyeq \bar{s_f}'$ with $\bar{\tau} \preccurlyeq \bar{\tau}'$. Therefore, $A \preccurlyeq \bar{s_t}' \curlyvee \bar{F} \curlywedge \bar{s_f}'$, and we conclude using rule $\textsc{Allow}_{\textsc{I}}$, $N = (\text{allowed}\ \bar{F}\ \text{then}\ \bar{N}_t\ \text{else}\ \bar{N}_f)$, $s = \bar{s_t}' \curlyvee \bar{F} \curlywedge \bar{s_f}'$ and $\tau' = \bar{\tau}'$.

□

The converse direction is not true, i.e. enforcement mechanism III accepts strictly more programs than enforcement mechanism II. This can be seen by considering the secure program where, $\theta_1 = \tau \xrightarrow{F_1} \sigma$ and $\theta_2 = \tau \xrightarrow{F_2} \sigma$:

$$(\text{if } (!\ a) \text{ then } (!\ (\text{ref}_{\theta_1}\ M_1)) \text{ else } (!\ (\text{ref}_{\theta_2}\ M_2))) \tag{12}$$

This program is not accepted by the type system of Section 4.3 because it cannot give the same type to both branches of the conditional (the type of the dereference of a reference of type θ is precisely θ). However, since the two types satisfy $\theta_1 \approx \theta_2$, the informative type system can accept it and give it the type $\theta_1 \curlywedge \theta_2$.

A more fundamental difference between the two enforcement mechanisms lays in the timing of the computation overhead that is required by each mechanism. While mechanism II requires heavy runtime type checks to occur each time a thread migrates, in III the typability analysis is anticipated to static time, leaving only a comparison between two flow policies to be performed at migration time. The complexity of this comparison depends on the concrete representation of flow policies. In the worst case, that of flow policies as general downward closure operators (see Section 2), it is linear on the number of security levels that are considered. When flow policies are flow relations, then it consists on a subset relation check, which is polynomial on the size of the flow policies.

4.4.4. Preservation of the semantics

We now prove that the program transformation that is encoded in the type system of Figure 7 preserves the semantics of the input expressions. To this end, we define a simulation between pools of threads written in the original language of Subsection 2.2, with pools of threads written in the language with annotations. The behavior of pools of threads of the former should be able to simulate step-by-step those of the latter, when operating on memories that are the "same", up to the annotations that distinguish the two languages. This intuitive concept is captured by means of the function annot() : $((\boldsymbol{Ref} \to \boldsymbol{Val}) \to \boldsymbol{Ref}) \to \boldsymbol{Val'}$), that extends the annotation process to stores:

$$\text{annot}(S)(a) = \hat{V} \ \textit{where}\ \exists \tau\ .\ \Gamma \vdash^{\Sigma} S(a) \hookrightarrow \hat{V} : \mho, \tau \tag{13}$$

The following simulation on pools of threads relates programs in the annotated language whose behavior exists already in the original (non-restricted) language.

Definition 4.14 ($\sim_\Gamma$)**.** *A Γ-simulation is a relation $\mathcal{S}$ on pools of threads, drawn from the original language and the annotated language, respectively, that satisfies, for all thread configurations T and for all memories $S, \hat{S}$ such that* $\text{annot}(S) = \hat{S}$*: $P_1\ \mathcal{S}\ P_2$ and $W \vdash \langle P_2, T, \hat{S}\rangle \xrightarrow[F]{d} \langle P_2', T', \hat{S}'\rangle$ implies:*

$$\exists P_1', S'\ .\ W \vdash \langle P_1, T, S\rangle \xrightarrow[F]{d} \langle P_1', T', S'\rangle \textit{ and } \text{annot}(S') = \hat{S}' \textit{ and } P_1'\ \mathcal{S}\ P_2'$$

The largest Γ-simulation is denoted by $\sim_\Gamma$.

For any Γ, the set of pairs of thread configurations where threads are values is a Γ-simulation. Furthermore, the union of a family of Γ-simulations is a Γ-simulation. Consequently, $\sim_\Gamma$ exists.

We can now check that the annotation process produces expressions that can be simulated in the above sense. In other words this means that enforcement mechanism III only enables behavior of programs that is already present in the original language.

Proposition 4.15. *Consider a given a typing environment* Γ *and reference labeling* Σ*. If there exist* s*,* τ *such that* $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$*, then for all thread names* $m \in \mathbf{Nam}$ *we have that* $\{M^m\} \sim_\Gamma \{N^m\}$*.*

Proof. We prove that the set

$$B = \{\langle \{M^m\}, \{N^m\}\rangle \mid m \in \mathbf{Nam} \text{ and } \exists s, \tau \,.\, \Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau\}$$

is a Γ-simulation acording to Definition 4.14. By case analysis on the proofs of $W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle P_2', T', S'\rangle$ and of $W \vdash \langle \{N^m\}, T, \hat{S}\rangle \xrightarrow[F]{d} \langle P_2', T', \hat{S}'\rangle$. □

To see that the behavior is restricted, it is enough to consider the program

$$(\text{thread}_l \ (\text{flow } \Omega \text{ in } (a := 1)) \text{ at } d)$$

which is transformed into $(\text{thread}_l^{\Omega} \ (\text{flow } \Omega \text{ in } (a := 1)) \text{ at } d)$ by the informative type system. This program blocks at the first execution step when $W(d) \neq \Omega$.

5. Related work

Information flow and distribution. The first approaches to the study of information flow in the presence of distribution do not consider distributed security settings. Mantel and Sabelfeld [29] provide a type system for preserving confidentiality for different kinds of channels established over a publicly observable medium in a distributed setting, but where interaction between domains is restricted to the exchange of values (no code mobility). Castagna, Bugliesi and Craffa study non-interference for a purely functional distributed and mobile calculus [17]; where no declassification mechanisms are contemplated. The work that is closest to this one is [4], which studies insecure information flows that are introduced by mobility in the context of a stateful distributed language that includes flow declarations. In the computation model that is considered, threads own references that move along with them during migration; this gives rise to *migration leaks* that result from memory synchronization issues.

Domains' security assurances can be differentiated as security levels. Zdancewic et. al [42] propose in Jif/Split a technique for automatically partitioning programs by placing code and data onto hosts in accordance with DLM labels [33] in the source code. Jif/Split ensures that if a host is subverted, the only data whose confidentiality or integrity is threatened during execution of a part of the program, is data owned by principals that trust that host. Chong et. al [16] present Swift as specialization of this idea for Web applications. Fournet et. al [20] present a compiler that produces distributed code where communications are implemented using cryptographic mechanisms, and ensures that all

confidentiality and integrity properties are preserved, despite the presence of active adversaries. In [43], Zheng and Myers address the issue of how availability of hosts might affect information flows in a distributed computation.

To our knowledge, the work presented in this paper (which includes [2, 5]) is the only one on information flow assuming distributed allowed flow policies. Most recently, the combination of the Non-disclosure for Networks Property and the Flow Policy Confinement Property for the same distributed security setting is shown to imply a generalization of Noninterference, referred to as the Distributed Noninterference property in [6].

Controlling declassification. Sabelfeld and Sands survey the literature regarding the subject of declassification [38], and observe that declassification can be controlled according to four main orthogonal goals as to: *what* information should be released [35, 26], *when* it should be allowed to happen [15], *who* should be authorized to use it [34], and *where* in the program it can be stated [14, 3]; these dimensions can also be combined [10]. Most of the overviewed approaches implicitly assume local settings, where the computation platform enforces fixed policies. Furthermore, the tools that are given to the programmer for controlling the usage of declassification operations are restricted to the declassifying operations themselves. In this sense, we can say that they provide for ways of controlling declassification *from within*, as opposed to the techniques that are proposed in this paper: both the allowed-construct and restricted versions of migration instructions are external to the flow declaration construct. In this sense, this paper advocates for technique for restricting declassification *from without*.

The concept of allowed flow policy as external to the program's own policies has been studied in simpler non-distributed contexts and in the absence of declassification [24, 8].

The work by Boudol and Kolundzija [13] on combining access control and declassification is the first to treat declassification control separately from the underlying information flow problem. In [13], standard access control primitives are used to control the access level of programs that perform declassifications in the setting of a local language, ensuring that a program can only declassify information that it has the right to read.

Controlling code mobility. A wide variety of distributed network models have been designed with the purpose of studying mechanisms for controlling code mobility. These range from type systems for statically controlling migration as an access control mechanism [30, 23], to runtime mechanisms that are based on the concept of programmable domain. In the latter, computing power is explicitly associated to the *membranes* of computation domains, and can be used for controlling boundary transposition. This control can be performed by processes that interact with external and internal programs [25, 39, 11], and can implement in particular runtime code migration enforcement mechanisms, or by more specific automatic verification mechanisms [22], for policies requiring different degree of expressiveness. In the present work we abstract away from

the particular machinery that implements the migration control checks, and express declaratively, via the language semantics, the condition that must be satisfied for the boundary transposition to be allowed.

Checking the validity of the declassification effect as a certificate is not simpler than checking the program against a concrete allowed policy (as presented in Subsection 4.3), meaning that it does not consist of a case of Proof Carrying Code. The concept of trust can be used to lift the checking requirements of code whose history of visited domains provides enough reassurance [22, 30]. These ideas could be applied to the present work, assisting the decision of trusting the declassification effect, otherwise leading to a full type check of the code.

Hybrid mechanisms. The use of hybrid mechanisms for enforcing information flow policies is currently an active research area (see [31] for a review of related work). The closest to ours is perhaps the study of securing information release for a simple language with dynamic code evaluation in the form of a string eval command, which includes an on-the-fly information flow static analysis [8].

Focusing on declassification control, the idea of using a notion of declassification effect for building a runtime migration control mechanism was put forward in [2] for a similar language with local thread creator and a basic goto migration instruction. In spite of the restrictions that are pointed out in Subsection 4.2 for a static analysis, the type system presented as part of enforcement mechanism I is more refined than the proof-of-concept presented earlier. Indeed, in the previous work, migration was not taken into account when analyzing the declassifications occurring within the migrating code. So while there the following program would be rejected if F was not allowed by $W(d_1)$

$$(\text{thread}_l\ (\text{thread}_l\ (\text{flow}\ F\ \text{in}\ M)\ \text{at}\ d_2)\ \text{at}\ d_1)$$

the type system of Figure 5 only rejects it if F is not allowed by $W(d_2)$. Enforcement mechanism II adopts part of the idea in [2] of performing a runtime type analysis to migrating programs, but uses a more permissive "checking" type system. Enforcement mechanism III explores a mechanism that allows to take advantage of the efficiency of flow policy comparisons. It uses a type and effect system for calculating declassification effects that is substantially more precise than previous ones, thanks to the matching relations and operations that it uses.

The concept of informative type and effect system was introduced in [7], where a different notion of declassification effect was defined and applied to the problem of dealing with dynamic updates to a local allowed flow policy.

6. Conclusions and Future work

The issue of controlling information flow in distributed settings with code mobility is a foundational problem in the field of Web security. In this work we have considered a simple network model that offers a distributed security setting, and adopted the standpoint of separating the enabling and controlling dimensions of declassification. We distinguish the definition and enforcement of two security properties that are related to declassification, namely Non-disclosure

for networks, which regards the compliance of information leaks to declassification declarations in the code, and Flow Policy Confinement, which regards compliance of those declarations to the allowed flow policies of each site. The proposed techniques are largely independent of the declassification mechanism that is used, which testifies on how a layer of control can be added to the most permissive declassification mechanisms. We expect that our framework can be easily adapted for studying other language constructs and settings, security properties and enforcement mechanisms:

- We have addressed this issue of ensuring that a thread can only migrate to a site if it complies to its allowed flow policy. One could also mention the dual problem, that information that is carried by programs into sites with more permissive flow policies becomes vulnerable. In order to tackle this problem, one could consider a model where references can move along with threads [4]. We leave this research direction for future work. Nevertheless, we believe that the allowed-condition construct that was introduced here can play an important role in the solution, since it enables threads to inspect the allowed flow policy of a site, according to which they can decide whether to remain there or to migrate away.

- When considering a strictly distributed memory model (where accesses to remote references are restricted), memory synchronization issues can lead to migration leaks as was shown in [4]. However, this paper shows that migration leaks do not exclusively depend on the memory model. In fact, even while assuming transparent remote accesses to references, a new form of migration leaks appear as a result of introducing our new program construction for inspecting the site's flow policies. This motivates a better understanding of migration leaks in global computations.

- The property of Flow Policy Confinement is perhaps the simplest form of imposing compliance of declassifications to distributed allowed flow policies. It would be interesting to consider properties with more complex (and restrictive) concerns, possibly taking into account the history of domains that a thread has visited when defining secure code migrations. For instance, one might want to forbid threads from moving to domains with more favorable allowed flow policies. The enforcement of such a property would be easily achieved by introducing a condition on the allowed flow policies of origin and destination domains. Using the abstractions in this paper, this could consist of the assumption $A \preccurlyeq W(d)$ to the migration typing rule, in order to restrict programs to migrate only to domains whose allowed flow policy is not more permissive than that that of the context where the thread is being created.

- We have considered an instance of the problem of enforcing compliance of declassifications to a dynamically changing allowed flow policy. In our setting, changes in the allowed flow policy result from the migration of programs during execution. We approach the problem from a migration

control perspective. To this end, we chose a network model that abstracts away the details of the migration control architecture. This allows us to prove soundness of a concrete network level security property, guaranteeing that programs can roam over the network, never performing declassifications that violate the network confinement property.

By performing comparisons between three related enforcement mechanisms, we have argued that the concept of declassification effect offers a good balance between precision and efficiency. We believe that similar mechanisms can be applied in other contexts. For future work, we plan to study others instances of enabling dynamic changing allowed flow policies.

References

[1] D. Akhawe, A. Barth, P. E. Lam, J. C. Mitchell, and D. Song. Towards a formal foundation of web security. In *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010*. IEEE Computer Society, 2010.

[2] A. Almeida Matos. Flow-policy awareness for distributed mobile code. In *Proceedings of CONCUR 2009 - Concurrency Theory*, volume 5710 of *Lecture Notes in Computer Science*. Springer, 2009.

[3] A. Almeida Matos and G. Boudol. On declassification and the non-disclosure policy. *Journal of Computer Security*, 17(5):549–597, 2009.

[4] A. Almeida Matos and J. Cederquist. Non-disclosure for distributed mobile code. *Mathematical Structures in Computer Science*, 21(6), 2011.

[5] A. Almeida Matos and J. Cederquist. Informative types and effects for hybrid migration control. In *Runtime Verification - 4th International Conference. Proceedings*, volume 8174 of *LNCS*. Springer, 2013.

[6] A. Almeida Matos and J. Cederquist. Distributed noninterference. In *Proceedings of the 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (1st special session 4PAD)*, 2014. to appear.

[7] A. Almeida Matos and J. Fragoso Santos. Typing illegal information flows as program effects. In *Proceedings of the 7th Workshop on Programming Languages and Analysis for Security*. ACM, 2012.

[8] A. Askarov and A. Sabelfeld. Tight enforcement of information-release policies for dynamic languages. In *Proceedings of the 2009 22nd IEEE Computer Security Foundations Symposium*, CSF '09, pages 43–59. IEEE Computer Society, 2009.

[9] G. Barthe, T. Rezk, and D. Naumann. Deriving an information flow checker and certifying compiler for java. In *SP '06: Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2006.

[10] G. Barthe, S. Cavadini, and T. Rezk. Tractable enforcement of declassification policies. In *CSF*, pages 83–97. IEEE Computer Society, 2008.

[11] G. Boudol. A generic membrane model. In Corrado Priami and Paola Quaglia, editors, *IST/FET International Workshop on Global Computing*, volume 3267 of *LNCS*, pages 208–222. Springer, 2005.

[12] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1–2):109–130, 2002.

[13] G. Boudol and M. Kolundzija. Access Control and Declassification. In *Computer Network Security*, volume 1 of *CCIS*, pages 85–98. Springer-Verlag, 2007.

[14] N. Broberg and D. Sands. Flow locks: Towards a core calculus for dynamic flow policies. In *Programming Languages and Systems. 15th European Symposium on Programming, ESOP 2006*, volume 3924 of *Lecture Notes in Computer Science*. Springer Verlag, 2006.

[15] S. Chong and A. C. Myers. Security policies for downgrading. In *Proc. of the 11th ACM conference on Computer and communications security*. ACM Press, 2004.

[16] S. Chong, J. Liu, A. Myers, X. Qi, K. Vikram, L. Zheng, and X. Zheng. Secure web applications via automatic partitioning. In *Proc. of 21st ACM Symposium on Operating Systems Principles*. ACM, 2007.

[17] S. Crafa, M. Bugliesi, and G. Castagna. Information flow security for boxed ambients. In Vladimiro Sassone, editor, *Workshop on Foundations of Wide Area Network Computing*, volume 66 of *ENTCS*, pages 76–97. Elsevier, 2002.

[18] D. E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[19] Riccardo Focardi and Roberto Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[20] C. Fournet, G. Le Guernic, and T. Rezk. A security-preserving compiler for distributed programs. In *Proc. of the 16th ACM Conf. on Computer and Communications Security*. ACM, 2009.

[21] J. A. Goguen and J. Meseguer. Security policies and security models. In *1982 IEEE Symp. on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982.

[22] D. Gorla, M. Hennessy, and V. Sassone. Security policies as membranes in systems for global computing. In *Foundations of Global Ubiquitous Computing, FGUC 2004*, ENTCS, pages 23–42. Elsevier, 2005.

[23] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In *Proc. of the 6th Int. Conf. on Foundations of Software Science and Computation Structures and European Conf. on Theory and Practice of Software*, pages 282–298. Springer-Verlag, 2003.

[24] M. Hicks, S. Tse, B. Hicks, and S. Zdancewic. Dynamic updating of information-flow policies. In *Workshop on Foundations of Comp. Security*, pages 7–18, 2005.

[25] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT ymposium on Principles of Programming Languages*, pages 352–364, New York, NY, USA, 2000. ACM.

[26] P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM Press, 2005.

[27] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *15th ACM Symp. on Principles of Programming Languages*, pages 47–57. ACM Press, 1988.

[28] H. G. Mairson. Deciding ml typability is complete for deterministic exponential time. In *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '90, pages 382–401. ACM, 1990.

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

[30] F. Martins and V.T. Vasconcelos. History-based access control for distributed processes. In *Proceedings of TGC'05*, LNCS. Springer-Verlag, 2005.

[31] S. Moore and S. Chong. Static analysis for efficient hybrid information-flow control. In *Proceedings of the 2011 IEEE 24th Computer Security Foundations Symposium*, CSF '11, pages 146–160. IEEE Computer Society, 2011.

[32] A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. In *19th IEEE Computer Society Symposium on Security and Privacy*, pages 186–197. IEEE Computer Society, 1998.

[33] A. C. Myers and B. Liskov. Protecting privacy using the decentralized label model. *ACM Trans. on Soft. Engineering and Methodology*, 9(4):410–442, 2000.

[34] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification and qualified robustness. *J. Comput. Secur.*, 14(2):157–196, 2006.

[35] A. Sabelfeld and A. Myers. A model for delimited information release. In *International Symposium on Software Security (ISSS'03)*, volume 3233 of *LNCS*. Springer-Verlag, 2004.

[36] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[37] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *CSFW'00: 13th IEEE Computer Security Foundations Workshop*, pages 200–215. IEEE Computer Society, 2000.

[38] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *18th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 2005.

[39] A. Schmitt and J.-B. Stefani. The m-calculus: a higher-order distributed process calculus. *SIGPLAN Not.*, 38(1):50–61, 2003.

[40] G. Smith. A new type system for secure information flow. In *CSFW'01: 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 2001.

[41] Andrew K. Wright and Matthias Felleisen. A syntactic approach to type soundness. *Informnation and Computation*, 115(1):38–94, 1994.

[42] S. Zdancewic, L. Zheng, N. Nystrom, and A. Myers. Secure program partitioning. *ACM Transactions on Computer Systems*, 20(3):283–328, 2002.

[43] L. Zheng and A. C. Myers. Making distributed computation trustworthy by construction. Technical Report TR2006-2040, Cornell Univ., 2006.

Appendix A. Proofs for "Controlling Information Flow"

Appendix A.1. Formalization of Non-Disclosure for Networks

Intuitively, Non-disclosure states that, at each computation step performed by a program, the information flow that occurs respects the flow policy (F) that is declared by the evaluation context where the command is executed. In [4] Non-disclosure is defined for Networks, considering a distributed setting with code mobility, by means of a bisimulation on pools of threads. In this paper we used a bisimulation on thread configurations. We prove that the new definition is weaker than the first.

Definition on pools of threads. We recall the definition in [4] here, using the notations of the current paper.

Definition Appendix A.1 ($\approx^{\Sigma,\Upsilon}_{\Gamma,l}$)**.** *Consider an allowed-policy mapping W, a reference labeling Σ, a thread labeling Υ, and a typing environment Γ. A $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation is a symmetric relation $\mathcal{R}$* ***on pools of threads*** *that satisfies, for all P_1, P_2, and for all (Σ,Γ)-compatible stores S_1, S_2:*

$$P_1 \ \mathcal{R}\ P_2 \text{ and } W \vdash \langle P_1, T_1, S_1\rangle \xrightarrow[\boldsymbol{F}]{d} \langle P'_1, T'_1, S'_1\rangle \text{ and } \langle T_1, S_1\rangle =^{\Sigma,\Upsilon}_{\boldsymbol{F},l} \langle T_2, S_2\rangle$$

with $\mathrm{dom}(S_1{}')-\mathrm{dom}(S_1) \cap \mathrm{dom}(S_2) = \emptyset$ *and* $\mathrm{dom}(T_1{}')-\mathrm{dom}(T_1) \cap \mathrm{dom}(T_2) = \emptyset$ *implies that there exist P'_2, T'_2, S'_2 such that:*

$$W \vdash \langle P_2, T_2, S_2\rangle \twoheadrightarrow \langle P'_2, T'_2, S'_2\rangle \text{ and } \langle T'_1, S'_1\rangle =^{\Sigma,\Upsilon}_{\mho,l} \langle T'_2, S'_2\rangle \text{ and } P'_1 \ \mathcal{R}\ P'_2$$

Furthermore, S'_1, S'_2 are still (W,Σ,Γ)-compatible.

For any Σ, Υ, Γ and l, the set of pairs of thread configurations where threads are values is an $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation. Furthermore, the union of a family of $(\Sigma,\Upsilon,\Gamma,l)$-bisimulations is a $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation. Consequently, $\approx^{\Sigma,\Upsilon,}_{\Gamma,l}$ exists.

Definition Appendix A.2 (Non-disclosure for Networks (on pools of threads))**.** *A pool of threads P satisfies the Non-disclosure for Networks property with respect to an allowed-policy mapping W, a reference labeling Σ, a thread labeling Υ and a typing environment Γ, if it satisfies $P \approx^{\Sigma,\Upsilon}_{\Gamma,l} P$ for all security levels l. We then write $P \in \mathcal{NDN}_1(W,\Sigma,\Upsilon,\Gamma)$.*

When imposing restrictions on the behaviors of related pools of threads, the above definition resets the position tracker arbitrarily at each step. Here (as in [4]), migration is subjective, meaning that only the thread itself can trigger its own migration. It is then reasonable to relax the power of the attacker, by focusing on the behavior of threads when coupled with their possible locations on the network.

Definition on thread configurations. We now present a weakened version of the property, that is defined over thread configurations.

Definition Appendix A.3 ($\approx^{\Sigma,\Upsilon}_{\Gamma,l}$)**.** *Consider an allowed-policy mapping W, a reference labeling Σ, and a typing environment Γ. A $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation is a symmetric relation $\mathcal{R}$* ***on thread configurations*** *that satisfies, for all P_1,T_1,P_2,T_2, and for all (Σ,Γ)-compatible memories S_1,S_2:*

$$\langle P_1,T_1\rangle\ \mathcal{R}\ \langle P_2,T_2\rangle \text{ and } W\vdash\langle P_1,T_1,S_1\rangle \xrightarrow[\boldsymbol{F}]{d}\langle P_1',T_1',S_1'\rangle \text{ and } \langle T_1,S_1\rangle =^{\Sigma,\Upsilon}_{\boldsymbol{F},l}\langle T_2,S_2\rangle$$

with $\mathrm{dom}(S_1{}')-\mathrm{dom}(S_1)\cap\mathrm{dom}(S_2)=\emptyset$ *and* $\mathrm{dom}(T_1{}')-\mathrm{dom}(T_1)\cap\mathrm{dom}(T_2)=\emptyset$ *implies that there exist P_2',T_2',S_2' such that:*

$$W\vdash\langle P_2,T_2,S_2\rangle\twoheadrightarrow\langle P_2',T_2',S_2'\rangle \text{ and } \langle T_1',S_1'\rangle =^{\Sigma,\Upsilon}_{\mho,l}\langle T_2',S_2'\rangle \text{ and } \langle P_1',T_1'\rangle\ \mathcal{R}\ \langle P_2',T_2'\rangle$$

Furthermore, S_1',S_2' are still (W,Σ,Γ)-compatible.

For any Σ, Υ, Γ and l, the set of pairs of thread configurations where threads are values is an $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation. Furthermore, the union of a family of $(\Sigma,\Upsilon,\Gamma,l)$-bisimulations is a $(\Sigma,\Upsilon,\Gamma,l)$-bisimulation. Consequently, $\approx^{\Sigma,\Upsilon,}_{\Gamma,l}$ exists.

Definition Appendix A.4 (Non-disclosure for Networks (on thread configurations))**.** *A pool of threads P satisfies the Non-disclosure for Networks property with respect to an allowed-policy mapping W, a reference labeling Σ, a thread labeling Υ and a typing environment Γ, if it satisfies $\langle P,T_1\rangle\approx^{\Sigma,\Upsilon}_{\Gamma,l}\langle P,T_2\rangle$ for all security level l and position trackers T_1,T_2 such that* $\mathrm{dom}(P)=\mathrm{dom}(T_1)=\mathrm{dom}(T_2)$ *and $T_1=^{\Sigma,\Upsilon}_{\mho,l}T_2$. We then write $P\in\mathcal{NDN}_2(W,\Sigma,\Upsilon,\Gamma)$.*

Comparison.

Proposition Appendix A.5. $\mathcal{NDN}_1(W,\Sigma,\Upsilon,\Gamma)\subseteq\mathcal{NDN}_2(W,\Sigma,\Upsilon,\Gamma)$.

Proof. We consider P in $\mathcal{NDN}_1(W,\Sigma,\Upsilon,\Gamma)$, i.e. such that for all security levels l we have $P\approx^{\Sigma,\Upsilon}_{\Gamma,l}P$ according to Definition Appendix A.1. Given any pair of position trackers T_1,T_2 such that $\mathrm{dom}(P)=\mathrm{dom}(T_1)=\mathrm{dom}(T_2)$ and $T_1=^{\Sigma,\Upsilon}_{\mho,l}T_2$, we prove that for all security levels l we have $\langle P,T_1\rangle\approx^{\Sigma,\Upsilon}_{\Gamma,l}\langle P,T_2\rangle$ according to Definition Appendix A.3. To this end, we consider the set

$$N=\{\langle\langle P_1,T_1\rangle,\langle P_2,T_2\rangle\rangle\mid \mathrm{dom}(P)=\mathrm{dom}(T_1)=\mathrm{dom}(T_2) \text{ and } T_1=^{\Sigma,\Upsilon}_{\mho,l}T_2 \text{ and } P_1\approx^{\Sigma,\Upsilon}_{\Gamma,l}P_2\}$$

and prove that $N\subseteq\approx^{\Sigma,\Upsilon}_{\Gamma,l}$ according to Definition Appendix A.3.

Assume that $\langle\langle P_1,T_1\rangle,\langle P_2,T_2\rangle\rangle\in N$, and suppose that for any given (Σ,Γ)-compatible memories S_1,S_2 we have $W\vdash\langle P_1,T_1,S_1\rangle\xrightarrow[\mathbf{F}]{d}\langle P_1',T_1',S_1'\rangle$ and

$\langle T_1, S_1\rangle =^{\Sigma,\Upsilon}_{\mathbf{F},l} \langle T_2, S_2\rangle$, with $\mathrm{dom}(S_1{}')-\mathrm{dom}(S_1) \cap \mathrm{dom}(S_2) = \emptyset$ and $\mathrm{dom}(T_1{}')-\mathrm{dom}(T_1) \cap \mathrm{dom}(T_2) = \emptyset$. Then, by Definition Appendix A.1 there exist P_2', T_2', S_2' such that

$$W \vdash \langle P_2, T_2, S_2\rangle \twoheadrightarrow \langle P_2', T_2', S_2'\rangle \text{ and } \langle T_1', S_1'\rangle =^{\Sigma,\Upsilon}_{\mho,l} \langle T_2', S_2'\rangle \text{ and } P_1' \mathcal{R} P_2'$$

Furthermore, S_1', S_2' are still (W, Σ, Γ)-compatible. It is now easy to see that $\langle\langle P_1', T_1'\rangle, \langle P_2', T_2'\rangle\rangle \in N$. □

Appendix A.2. Type system

Appendix A.2.1. Subject Reduction

In order to establish the soundness of the type system of Figure 4 we need a Subject Reduction result, stating that types that are given to expressions are preserved by computation. To prove it we follow the usual steps [41].

Remark Appendix A.6.

1. *If* $W \in \boldsymbol{Pse}$ *and* $\Gamma \vdash^{\Sigma}_{j,F} W : s, \tau$*, then for all domain names* d'*, security levels* j'*, flow policies* F' *and security effects* s'*, we have that* $\Gamma \vdash^{\Sigma}_{j',F'} W : s', \tau$.
2. *For any flow policies* F, F'*, such that* $F' \preccurlyeq F$*, we have that* $\Gamma \vdash^{\Sigma}_{j,F} M : \tau$ *implies* $\Gamma \vdash^{\Sigma}_{j,F'} M : \tau$.

Lemma Appendix A.7.

1. *If* $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$ *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} M : s, \tau$.
2. *If* $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} M : s, \tau$ *and* $x \notin \mathrm{fv}(M)$ *then* $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$.

Proof. By induction on the inference of the type judgment. □

Lemma Appendix A.8 (Substitution)**.**
If $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} M : s, \tau$ *and* $\Gamma \vdash^{\Sigma}_{j',F'} W : s', \sigma$ *then* $\Gamma \vdash^{\Sigma}_{j,F} \{x \mapsto W\}M : s, \tau$.

Proof. By induction on the inference of $\Gamma, x : \tau \vdash^{\Sigma}_{j,F} M : s, \sigma$, and by case analysis on the last rule used in this typing proof, using the previous lemma. Let us examine the cases related to the new language constructs:

Mig. Here $M = (\mathrm{thread}_l\ \bar{M}\ \mathrm{at}\ \bar{d})$ and we have that $\Gamma, x : \sigma \vdash^{\Sigma}_{l,\mho} \bar{M} : \bar{s}, \tau$, with $\tau = \mathsf{unit}$ and $\bar{s} = \langle \bot, l \sqcup s.w, \bot\rangle$. By induction hypothesis, then $\Gamma \vdash^{\Sigma}_{l,\mho} \{x \mapsto W\}\bar{M} : \bar{s}, \tau$. Therefore, by rule MIG, $\Gamma \vdash^{\Sigma}_{j,F} (\mathrm{thread}_l\ \{x \mapsto W\}\bar{M}\ \mathrm{at}\ \bar{d}) : s, \tau$.

Flow. Here $M = (\mathrm{flow}\ \bar{F}\ \mathrm{in}\ \bar{M})$, $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F \curlywedge \bar{F}} \bar{M} : s, \tau$. By induction hypothesis, $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \bar{F}} \{x \mapsto W\}\bar{M} : s, \tau$. Then, by FLOW, we have $\Gamma \vdash^{\Sigma}_{j,F} (\mathrm{flow}\ \bar{F}\ \mathrm{in}\ \{x \mapsto W\}\bar{M}) : s, \tau$.

Allow. Here $M = (\text{allowed } \bar{F} \text{ then } N_t \text{ else } N_f)$ and we have $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} N_t : s_t, \tau$ and $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} N_f : s_f, \tau$ with $j \sqsubseteq^F s_t.w, s_f.w$ and $s = s_t \sqcup s_f \sqcup \langle \bot, \top, j \rangle$. By induction hypothesis, $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} \{x \mapsto W\}N_t : s_t, \tau$ and $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} \{x \mapsto W\}N_f : s_f, \tau$. Therefore, by rule ALLOW, we have that $\Gamma, x : \sigma \vdash^{\Sigma}_{j,F} (\text{allowed } \bar{F} \text{ then } \{x \mapsto W\}N_t \text{ else } \{x \mapsto W\}N_f) : s, \tau$.

□

Lemma Appendix A.9 (Replacement)**.**
If $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[M] : s, \tau$ *is a valid judgment, then the proof gives* M *a typing* $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \mathrm{E} \rceil} M : \bar{s}, \bar{\tau}$ *for some* $\bar{s}$ *and* $\bar{\tau}$ *such that* $\bar{s} \sqsubseteq s$*. In this case, if* $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \mathrm{E} \rceil} N : \bar{s}', \bar{\tau}$ *with* $\bar{s}' \sqsubseteq \bar{s}$*, then* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[N] : s', \tau$*, for some* s' *such that* $s' \sqsubseteq s$*.*

Proof. By induction on the structure of E. Let us examine the case of the flow declaration, which is the only non-standard evaluation context:

E[*M*] = (flow *F′* in $\bar{\mathrm{E}}$[*M*]). By FLOW, we have $\Gamma \vdash^{\Sigma}_{j,F \curlywedge F'} \bar{\mathrm{E}}[M] : s, \tau$. By induction hypothesis, the proof gives M a typing $\Gamma \vdash^{\Sigma}_{j,F \curlywedge F' \curlywedge \lceil \bar{\mathrm{E}} \rceil} M : \hat{s}, \hat{\tau}$, for $\hat{s}, \hat{\tau}$ such that $\hat{s} \sqsubseteq s$. Also by induction hypothesis, $\Gamma \vdash^{\Sigma}_{j,F \curlywedge F'} \bar{\mathrm{E}}[N] : s', \tau$, for some s' such that $s' \sqsubseteq s$. Then, again by FLOW, we have $\Gamma \vdash^{\Sigma}_{j,F}$ (flow F' in $\bar{\mathrm{E}}[N]) : s', \tau$.

□

We check that the type of a thread is preserved by reduction, while its effects "weaken".

Proposition Appendix A.10 (Subject Reduction)**.** *Given a reference and thread labeling* Σ*,* Υ*, consider a thread* M^m *for which there exist* Γ*,* F*,* s *and* τ *such that* $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$ *and suppose* $W \vdash \langle \{M^m\}, T, S \rangle \xrightarrow[F']{d} \langle \{M'^m\} \cup P, T', S' \rangle$*, for a memory* S *that is* (Σ, Γ)*-compatible. Then, there is an effect* s' *such that* $s' \sqsubseteq s$ *and* $\Gamma \vdash^{\Sigma}_{j,F} M' : s', \tau$*, and* S' *is also* (Σ, Γ)*-compatible. Furthermore, if* $P = \{N^n\}$*, for some expression* N *and thread name* n*, then there exists* s'' *such that* $s.w \sqsubseteq s''.w$ *such that* $\Gamma \vdash^{\Sigma}_{\Upsilon(k),\mathcal{U}} N : s'', \mathsf{unit}$*.*

Proof. Suppose that $M = \bar{\mathrm{E}}[\bar{M}]$ and $W \vdash \langle \{\bar{M}^m\}, T, S \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{M}'^m\} \cup P', \bar{T}', \bar{S}' \rangle$. We start by observing that this implies $F = \bar{F} \curlywedge \lceil \bar{\mathrm{E}} \rceil$, $M' = \bar{\mathrm{E}}[\bar{M}']$, $P = P'$, $\bar{T}' = T'$ and $\bar{S}' = S'$. We can assume, without loss of generality, that $\bar{M}$ is the smallest in the sense that there is no $\hat{\mathrm{E}}, \hat{M}, \hat{N}$ such that $\hat{\mathrm{E}} \neq []$ and $\hat{\mathrm{E}}[\hat{M}] = \bar{M}$ for which we can write $W \vdash \langle \{\hat{M}^m\}, T, S \rangle \xrightarrow[\hat{F}]{d} \langle \{\hat{M}'^m\} \cup P, T', S' \rangle$.

By Lemma Appendix A.9, we have $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \bar{\mathrm{E}} \rceil} \bar{M} : \bar{s}, \bar{\tau}$ in the proof of $\Gamma \vdash^{\Sigma}_{j,F} \bar{\mathrm{E}}[\bar{M}] : s, \tau$, for some $\bar{s}$ and $\bar{\tau}$. We proceed by case analysis on the transition $W \vdash \langle \{\bar{M}^m\}, T, S \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{M}'^m\} \cup P, T', S' \rangle$, and prove that if $S' \neq S$ then:

- There is an effect $\bar{s}'$ such that $\bar{s}' \sqsubseteq \bar{s}$ and $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \bar{\mathrm{E}} \rceil} \bar{M}' : \bar{s}', \bar{\tau}$. Furthermore, for every reference $a \in \mathrm{dom}(S')$ implies $\Gamma \vdash^{\Sigma}_{j,\mho} S'(a) : \bot, \Sigma_2(a)$.
- If $P = \{N^n\}$ for some expression N and thread name n, then there is an effect $\bar{s}''$ such that $\bar{s}.w \sqsubseteq \bar{s}''.w$ and a thread name d' such that $\Gamma \vdash^{\Sigma}_{\Upsilon(n),\mho} N : \bar{s}'', \mathsf{unit}$.

$\bar{M}$ = (allowed F' then N_t else N_f). Suppose that $W(d) \preccurlyeq F'$ (the other case is analogous). Here we have $\bar{M}' = N_t$, $S = S'$ and $P = \emptyset$. By Allow, we have that $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \bar{\mathrm{E}} \rceil} N_t : s_t, \bar{\tau}$, where $s_t \sqsubseteq \bar{s}$.

$\bar{M}$ = (flow F' in V). Here we have $\bar{M}' = V$, $S = S'$ and $P = \emptyset$. By rule Flow, we have that $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge F'} V : \bar{s}, \tau.$ and by Remark Appendix A.6, we have $\Gamma \vdash^{\Sigma}_{j,F \curlywedge \lceil \bar{\mathrm{E}} \rceil} V : \bar{s}, \bar{\tau}$.

$\bar{M}$ = (thread$_k$ N at d'). Here we have $\bar{M}' = ()$, $P = \{N^n\}$ for some thread name n $S = S'$ and $T'(n) = d$. By Mig, we have that $\Gamma \vdash^{\Sigma}_{\Upsilon(n),\mho} N : \hat{s}, \mathsf{unit}$, with $s.w \sqsubseteq \hat{s}.w$ and $\bar{\tau} = \mathsf{unit}$, and by Nil we have that $\Gamma \vdash^{\Sigma}_{F \curlywedge \lceil \bar{\mathrm{E}} \rceil,} () : \bar{s}, \mathsf{unit}$.

By Lemma Appendix A.9, we can finally conclude that $\Gamma \vdash^{\Sigma}_{j,F} \bar{\mathrm{E}}[\bar{M}'] : s', \tau$, for some $s' \sqsubseteq s$. □

Appendix A.2.2. Basic Properties

Properties of the Semantics. One can prove that the semantics preserves the conditions for well-formedness, and that a configuration with a single expression has at most one transition, up to the choice of new names.

The following result states that, if the evaluation of a thread M^m differs in the context of two distinct states while not creating two distinct reference names or thread names, this is because either M^m is performing a dereferencing operation, which yields different results depending on the memory, or because M^m is testing the allowed policy.

Lemma Appendix A.11 (Splitting Computations)**.**
If we have $W \vdash \langle \{M^m\}, T_1, S_1 \rangle \xrightarrow[F]{d} \langle P_1', T_1', S_1' \rangle$ *and* $W \vdash \langle \{M^m\}, T_2, S_2 \rangle \xrightarrow[F']{d} \langle P_2', T_2', S_2' \rangle$ *with* ${P_1}' \neq {P_2}'$*, then* $P_1' = \{{M_1}'^m\}$*,* $P_2' = \{{M_2}'^m\}$ *and either:*

- $\exists \mathrm{E}, a$ *such that* $F = \lceil \mathrm{E} \rceil = F'$*,* $M = \mathrm{E}[(!\ a)]$*, and* $M_1' = \mathrm{E}[S_1(a)]$*,* $M_2' = \mathrm{E}[S_2(a)]$ *with* $\langle T_1', S_1' \rangle = \langle T_1, S_1 \rangle$ *and* $\langle T_2', S_2' \rangle = \langle T_2, S_2 \rangle$*, or*
- $\exists \mathrm{E}, \bar{F}$ *such that* $F = \lceil \mathrm{E} \rceil = \bar{F}$*,* $M = \mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$*, and* $T_1(m) \neq T_2(m)$ *with* $\langle T_1', S_1' \rangle = \langle T_1, S_1 \rangle$ *and* $\langle T_2', S_2' \rangle = \langle T_2, S_2 \rangle$*.*

Proof. Note that the only rules that depend on the state are those for the reduction of $\mathrm{E}[(!\ a)]$ and of $\mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$. By case analysis on the transition $W \vdash \langle \{M^m\}, T_1, S_1 \rangle \xrightarrow[F]{d} \langle P_1', T_1', S_1' \rangle$. □

Effects.

Lemma Appendix A.12 (Update of Effects)**.**

1. *If* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(!\ a)] : s, \tau$ *then* $\Sigma_1(a) \sqsubseteq s.r$.
2. *If* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(a := V)] : s, \tau$, *then* $s.w \sqsubseteq \Sigma_1(a)$.
3. *If* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(\mathrm{ref}_{l,\theta}\ V)] : s, \tau$, *then* $s.w \sqsubseteq l$.
4. *If* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(\mathrm{thread}_l\ M \text{ at } d')] : s, \tau$, *then* $s.w \sqsubseteq l$.
5. *If* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(\text{allowed } F \text{ then } N_t \text{ else } N_f)] : s, \tau$, *then* $j \sqsubseteq s.t$.

Proof. By induction on the structure of E. □

High Expressions. We can identify a class of threads that have the property of never performing any change in the "low" part of the memory. These are classified as being "high" according to their behavior[2]:

Definition Appendix A.13 (Operationally High Threads)**.** *A set of threads* $\mathcal{H}^{\Sigma,\Upsilon}$ *is a set of* operationally (Σ, Υ, F, l)-high threads *if the following holds for all* $M^m \in \mathcal{H}^{\Sigma,\Upsilon}$, *for all states* $\langle T, S\rangle$, *and for all flow policy mappings* W*:*

$$W \vdash \langle \{M^m\}, T, S\rangle \xrightarrow[F']{d} \langle P', T', S'\rangle \text{ implies } \langle T, S\rangle =^{\Sigma,\Upsilon}_{F,l} \langle T', S'\rangle \text{ and } P' \subseteq \mathcal{H}^{\Sigma,\Upsilon}$$

The largest set of operationally (Σ, Υ, F, l)*-high threads is denoted by* $\mathcal{H}^{\Sigma,\Upsilon}_{F,l}$. *We then say that a thread* M^m *is operationally* (Σ, Υ, F, l)*-high, if* $M^m \in \mathcal{H}^{\Sigma_1,\Upsilon}_{F,l}$.

Remark that for any F and l there exists a set of operationally (F, l)-high threads, like for instance $\{V^m \mid V \in \boldsymbol{Val}\}$. Furthermore, the union of a family of sets of operationally (F, l)-high threads is a set of operationally (F, l)-high threads. Notice that if $F' \subseteq F$, then any operationally (Σ, Υ, F, l)-high thread is also operationally $(\Sigma, \Upsilon, F', l)$-high.

Some expressions can be easily classified as "high" by the type system, which only considers their syntax. These cannot perform changes to the "low" memory simply because their code does not contain any instruction that could perform them. Since the writing effect is intended to be a lower bound to the level of the references that the expression can create or assign to, expressions with a high writing effect can be said to be *syntactically high*:

Definition Appendix A.14 (Syntactically High Expressions)**.** *An expression* M *is syntactically* $(\Sigma, \Gamma, j, F, l)$*-high if there exists* s, τ *such that* $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$ *with* $s.w \not\sqsubseteq^F l$. *The expression* M *is a syntactically* $(\Sigma, \Gamma, j, F, l)$*-high function if there exists* j', F', s, τ, σ *such that* $\Gamma \vdash^{\Sigma}_{j',F'} M : \tau \xrightarrow[j,F]{s} \sigma$ *with* $s.w \not\sqsubseteq^F l$.

[2]The notion of "operationally high thread" that we define here should not not be confused with the notion of "high thread". The former refers to the security level that is associated with a thread, while the latter refers to the changes that the thread performs on the state.

We can now state that syntactically high expressions have an operationally high behavior.

Lemma Appendix A.15 (High Expressions)**.** *If M is a syntactically (Σ,Γ,j,F,l)-high expression, and $\Upsilon(m)=j$, then M^m is an operationally (Σ,Υ,F,l)-high thread.*

Proof. We show that the set

$$\{M^m \mid \exists j\ .\ M \textit{ is syntactically } (\Sigma,\Gamma,j,F,l)\textit{-high}\}$$

is a set of operationally (Σ,Υ,F,l)-high threads, i.e.: if M is syntactically (Σ,Γ,j,F,l)-high, that is if there exists s,τ such that $\Gamma \vdash^{\Sigma}_{j,F} M : s,\tau$ with $s.w \not\sqsubseteq^F l$, and, for some policy mapping W, if $W \vdash \langle\{M^m\},T,S\rangle \xrightarrow[F']{d} \langle\{M'^m\}\cup P,T',S'\rangle$ then $\langle T,S\rangle =^{\Sigma_1,\Upsilon}_{F,l} \langle T',S'\rangle$. This is enough since, by Subject Reduction (Theorem Appendix A.10), M' is syntactically (Σ,Γ,j,F,l)-high, and if $P=\{N^n\}$ for some expression N and thread name n, then by Remark Appendix A.6 also N is syntactically (Σ,Γ,k,F,l)-high for some k. We proceed by cases on the proof of the transition $W \vdash \langle\{M^m\},T,S\rangle \xrightarrow[F']{d} \langle\{M'^m\},T',S'\rangle$. The lemma is trivial in all the cases where $\langle T,S\rangle = \langle T',S'\rangle$.

***M* = E[(thread$_k$ *N* at *d′*)].** Here $P=\{N^n\}$ for some thread name n, $S'=S$ and $T'=[n:=d']T$. By Lemma Appendix A.12, $s.w \sqsubseteq k$. This implies $k \not\sqsubseteq^F l$, and by assumption $k=\Upsilon(n) \not\sqsubseteq^F l$, hence $T' =^{\Sigma_1,\Upsilon}_{F,l} T$.

□

Appendix A.2.3. Soundness

Behavior of "Low"-Terminating Expressions. Recall that, according to the intended meaning of the termination effect, the termination or non-termination of expressions with low termination effect should only depend on the low part of the state. In other words, two computations of a same thread running under two "low"-equal states should either both terminate or both diverge. In particular, this implies that termination-behavior of these expressions cannot be used to leak "high" information when composed with other expressions (via termination leaks).

The following guaranteed-transition result holds for low-equal states.

Lemma Appendix A.16 (Guaranteed Transitions)**.** *Consider a flow policy mapping W, a thread M^m and two states $\langle T_1,S_1\rangle,\langle T_2,S_2\rangle$ such that $W \vdash \langle\{M^m\},T_1,S_1\rangle \xrightarrow[F]{d} \langle P_1',T_1',S_1'\rangle$, and for some F' we have $\langle T_1,S_1\rangle =^{\Sigma_1,\Upsilon}_{F\curlywedge F',low} \langle T_2,S_2\rangle$. Then:*

- *If $P_1'=\{M_1'^m\}$, and $a \in \mathrm{dom}(S_1'-S_1)$ implies that a is fresh for S_2, then there exist M_2', T_2' and S_2' such that $W \vdash \langle\{M^m\},T_2,S_2\rangle \xrightarrow[F]{d} \langle\{M_2'^m\},T_2',S_2'\rangle$ and $\langle T_1',S_1'\rangle =^{\Sigma_1,\Upsilon}_{F\curlywedge F',low} \langle T_2',S_2'\rangle$.*

- *If* $P_1' = \{M'^m, N^n\}$ *for some expression* N *and thread name* $n \notin \mathrm{dom}(T_2)$*, then there exist* M_2'*,* T_2' *and* S_2' *such that we have* $W \vdash \langle \{M^m\}, T_2, S_2\rangle \xrightarrow[F]{d} \langle \{M'^m, N^n\}, T_2', S_2'\rangle$ *and* $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2', S_2'\rangle$*.*

Proof. By case analysis on the proof of $W \vdash \langle \{M^m\}, T_1, S_1\rangle \xrightarrow[F]{d} \langle P_1', T_1', S_1'\rangle$. In most cases, this transition does not modify or depend on the state $\langle T_1, S_1\rangle$, and we may let $P_2' = P_1'$ and $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$.

$\boldsymbol{M_1 = \mathbf{E}[(\textbf{allowed } F \textbf{ then } N_t \textbf{ else } N_f)] \textbf{ and } W(T_1(m)) \preccurlyeq F.}$ Here $P_1' = \{\mathrm{E}[N_t]^m\}$, $F = \lceil \mathrm{E} \rceil$, and $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$. There are two possible cases:

- If $W(T_2(m)) \preccurlyeq F$, then $W \vdash \langle \{M^m\}, T_2, S_2\rangle \xrightarrow[F]{d} \langle \{{N_t}^m\}, T_2, S_2\rangle$.
- If $W(T_2(m)) \not\preccurlyeq F$, then $W \vdash \langle \{M^m\}, T_2, S_2\rangle \xrightarrow[F]{d} \langle \{{N_f}^m\}, T_2, S_2\rangle$.

Clearly, in both cases, By assumption, $\langle T_1, S_1\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2, S_2\rangle$.

$\boldsymbol{M_1 = \mathbf{E}[(\textbf{thread}_k\ N \textbf{ at } d')].}$ Here, for a thread name n, $P_1' = \{\mathrm{E}[()]^m, N^n\}$, $F = \lceil \mathrm{E} \rceil$, $T_1' = T_1 \cup \{n \mapsto d'\}$, and $S_1' = S_1$. Assuming that $n \notin \mathrm{dom}(T_2)$, we also have that $W \vdash \langle \{M^m\}, T_2, S_2\rangle \xrightarrow[F]{d} \langle \{\mathrm{E}[()]^m, N^n\}, T_2 \cup \{n \mapsto d'\}, S_2\rangle$. Clearly, $\langle T_1 \cup \{n \mapsto d'\}, S_1\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2 \cup \{n \mapsto d'\}, S_2\rangle$.

□

We aim at proving that any typable thread M^m that has a low-termination effect always presents the same behavior according to a *strong* bisimulation on low-equal states: if two continuations M_1^m and M_2^m of M^m are related, and if M_1^m can perform an execution step over a certain state, then M_2^m can perform the same low changes to any low-equal state in precisely one step, while the two resulting continuations are still related. This implies that any two computations of M^m under low-equal states should have the same "length", and in particular they are either both finite or both infinite. To this end, we design a reflexive binary relation on expressions with low-termination effects that is closed under the transitions of Guaranteed Transitions (Lemma Appendix A.16).

The inductive definition of $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ is given in Figure A.8. Notice that it is a symmetric relation. In order to ensure that expressions that are related by $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ perform the same changes to the low memory, its definition requires that the references that are created or written using (potentially) different values are high.

Remark Appendix A.18. *If for* Σ*,* Γ*,* j*,* F *and low we have* $M_1\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ M_2$ *and* $M_1 \in \boldsymbol{Val}$*, then* $M_2 \in \boldsymbol{Val}$*.*

From the following lemma one can conclude that the relation $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ relates the possible outcomes of expressions that are typable with a low termination effect, and that perform a high read over low-equal memories.

Definition Appendix A.17 ($\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$)**.** *We have that* $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ *if* $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1, \tau$ *and* $\Gamma \vdash^{\Sigma}_{j,F} M_2 : s_2, \tau$ *for some* s_1, s_2 *and* τ *with* $s_1.t \sqsubseteq^F low$ *and* $s_2.t \sqsubseteq^F low$ *and one of the following holds:*

Clause 1. M_1 *and* M_2 *are both values, or*

Clause 2. $M_1 = M_2$*, or*

Clause 3. $M_1 = (\bar{M_1}; \bar{N})$ *and* $M_2 = (\bar{M_2}; \bar{N})$ *where* $\bar{M_1} \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M_2}$*, or*

Clause 4. $M_1 = (\mathrm{ref}_{l,\theta} \ \bar{M_1})$ *and* $M_2 = (\mathrm{ref}_{l,\theta} \ \bar{M_2})$ *where* $\bar{M_1} \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M_2}$*, and* $l \not\sqsubseteq^F low$*, or*

Clause 5. $M_1 = (! \ \bar{M_1})$ *and* $M_2 = (! \ \bar{M_2})$ *where* $\bar{M_1} \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M_2}$*, or*

Clause 6. $M_1 = (\bar{M_1} := \bar{N_1})$ *and* $M_2 = (\bar{M_2} := \bar{N_2})$ *with* $\bar{M_1} \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{M_2}$*, and* $\bar{N_1} \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{N_2}$*, and* $\bar{M_1}, \bar{M_2}$ *both have type* $\theta \ \mathrm{ref}_l$ *for some* θ *and* l *such that* $l \not\sqsubseteq^F low$*, or*

Clause 7. $M_1 = (\text{flow } F' \text{ in } \bar{M_1})$ *and* $M_2 = (\text{flow } F' \text{ in } \bar{M_2})$ *with* $\bar{M_1} \ \mathcal{T}^{\Sigma,\Gamma}_{j,F \curlywedge F',low} \ \bar{M_2}$*.*

Figure A.8: The relation $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$

Lemma Appendix A.19. *If for some Σ, Γ, j and F there exist s, τ such that $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(!\ a)] : s, \tau$ with $s.t \sqsubseteq^F$ low and $\Sigma_1(a) \not\sqsubseteq_{F \curlywedge \lceil \mathrm{E} \rceil}$ low, then for any values $V_0, V_1 \in$ **Val** such that $\Gamma \vdash^{\Sigma}_{j,\mathfrak{V}} V_i : \bot, \theta$ we have $\mathrm{E}[V_0]\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ \mathrm{E}[V_1]$.*

Proof. By induction on the structure of E.

$\mathbf{E}[(!\ a)] = (\mathbf{flow}\ F'\ \mathbf{in}\ \mathbf{E_1}[(!\ a)])$. By rule FLOW we have $\Gamma \vdash^{\Sigma}_{j,F \curlywedge F'} V : s, \tau$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F \curlywedge F', low}\ \mathrm{E}_1[V_1]$, so we conclude by Lemma Appendix A.9 and Clause 7. Therefore $\bar{s}.t \sqsubseteq^F low$, and since $\Sigma_1(a) \not\sqsubseteq^{F \curlywedge \mathrm{E}} low$ implies $\Sigma_1(a) \not\sqsubseteq^{F \curlywedge \mathrm{E}_1} low$, then by induction hypothesis we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ \mathrm{E}_1[V_1]$. By Lemma Appendix A.9 and Clause 8 we can conclude.

□

We can now prove that $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ behaves as a kind of "strong bisimulation":

Proposition Appendix A.20 (Strong Bisimulation for Low-Termination)**.** *If we have $M_1\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ M_2$ and, for a given allowed flow policy mapping W, also $W \vdash \langle \{{M_1}^m\}, T_1, S_1\rangle \xrightarrow[F']{d} \langle P_1', T_1', S_1'\rangle$, with $\Upsilon(m) = j$ and $\langle T_1, S_1\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2, S_2\rangle$, then:*

- *If $P_1' = \{{M_1'}^m\}$, and $a \in \mathrm{dom}(S_1' - S_1)$ implies that a is fresh for S_2, then there exist M_2', T_2' and S_2' such that $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m\}, T_2', S_2'\rangle$ with $M_1'\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ M_2'$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2', S_2'\rangle$.*
- *If $P_1' = \{{M_1'}^m, N^n\}$ for some expression N and thread name $n \notin \mathrm{dom}(T_2)$, then there exist M_2', T_2' and S_2' such that $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m, N^n\}, T_2', S_2'\rangle$ with $M_1'\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ M_2'$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2', S_2'\rangle$.*

Proof. By case analysis on the clause by which $M_1\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ M_2$, and by induction on the definition of $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$. In the following, we use Subject Reduction (Theorem Appendix A.10) to guarantee that the termination effect of the expressions resulting from M_1 and M_2 is still low with respect to *low* and F. This, as well as typability (with the same type) for j, F and *low*, is a requirement for being in the $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ relation.

Clause 2. Here $M_1 = M_2$. By Guaranteed Transitions (Lemma Appendix A.16), then:

- If $P_1' = \{{M_1'}^m\}$, and $a \in \mathrm{dom}(S_1' - S_1)$ implies that a is fresh for S_2, then there exist M_2', T_2' and S_2' such that $W \vdash \langle \{M^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m\}, T_2', S_2'\rangle$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2', S_2'\rangle$. There are two cases to consider:

$M_2' = M_1'$. Then we have $M_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$, by Clause 2 and Subject Reduction (Theorem Appendix A.10).

$M_2' \neq M_1'$. Then by Splitting Computations (Lemma Appendix A.11) we have two possibilities:
(1) there exist E and a such that $M_1' = \mathrm{E}[S_1(a)]$, $F' = \lceil \mathrm{E} \rceil$, $M_2' = \mathrm{E}[S_2(a)]$, $\langle T_1', S_1' \rangle = \langle T_1, S_1 \rangle$ and $\langle T_2', S_2' \rangle = \langle T_2, S_2 \rangle$. Since $S_1(a) \neq S_2(a)$, we have $\Sigma_1(a) \not\sqsubseteq^{F \curlywedge F'} low$. Therefore, $M_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$, by Lemma Appendix A.19 above.
(2) there exists E such that $M_1' = \mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$, $F' = \lceil \mathrm{E} \rceil$, and $T_1(m) \neq T_2(m)$ with $\langle T_1', S_1' \rangle = \langle T_1, S_1 \rangle$ and $\langle T_2', S_2' \rangle = \langle T_2, S_2 \rangle$. Since $T_1(m) \neq T_2(m)$, we have $\Upsilon(m) = j \not\sqsubseteq^F low$, and by Lemma Appendix A.12 we have $j \sqsubseteq^F s.t$, so $s.t \not\sqsubseteq^F low$, which contradicts the assumption.

- If $P_1' = \{M_1'^m, N^n\}$ for some expression N and thread name $n \notin \mathrm{dom}(T_2)$, then there exist M_2', T_2' and S_2' s.t. $W \vdash \langle \{M^m\}, T_2, S_2 \rangle \xrightarrow[F']{d} \langle \{M_2', N^n\}, T_2', S_2' \rangle$ and $\langle T_1', S_1' \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2', S_2' \rangle$. By Splitting Computations (Lemma Appendix A.11), necessarily $M_1'^m = M_2'^m$. Then we have $M_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$, by Clause 2 and Subject Reduction (Theorem Appendix A.10).

Clause 7. Here we have $M_1 = (\text{flow } \bar{F} \text{ in } \bar{M}_1)$ and $M_2 = (\text{flow } \bar{F} \text{ in } \bar{M}_2)$ and $\bar{M}_1 \; \mathcal{T}^{\Sigma,\Gamma}_{j,F \curlywedge \bar{F},low} \; \bar{M}_2$. There are two cases.

$\bar{M}_1$ can compute. In this case we have $M_1' = (\text{flow } \bar{F} \text{ in } \bar{M}_1')$ with $W \vdash \langle \{\bar{M}_1^m\}, T_1, S_1 \rangle \xrightarrow[F'']{d} \langle \{\bar{M}_1'^m\}, T_1', S_1' \rangle$ with $F' = \bar{F} \curlywedge F''$. To use the induction hypothesis, there are three possible cases:

- If $\bar{P}_1' = \{\bar{M}_1'^m\}$, and $a \in \mathrm{dom}(S_1' - S_1)$ implies that a is fresh for S_2, then there exist $\bar{M}_2'$, T_2' and S_2' s.t. $W \vdash \langle \{\bar{M}_2^m\}, T_2, S_2 \rangle \xrightarrow[F'']{d} \langle \{\bar{M}_2'^m\}, T_2', S_2' \rangle$ with $\bar{M}_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F \curlywedge \bar{F},low} \; \bar{M}_2'$ and $\langle T_1', S_1' \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F' \curlywedge \bar{F},low} \langle T_2', S_2' \rangle$. Notice that $\langle T_1', S_1' \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2' \rangle$.
- If $\bar{P}_1' = \{\bar{M}_1'^m, N^n\}$ for some expression N and thread name $n \notin \mathrm{dom}(T_2)$, then there exist $\bar{M}_2'$, T_2' and S_2' such that $W \vdash \langle \{\bar{M}_2^m\}, T_2, S_2 \rangle \xrightarrow[F'']{d} \langle \{\bar{M}_2'^m, N^n\}, T_2', S_2' \rangle$ with $\bar{M}_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F \curlywedge \bar{F},low} \; \bar{M}_2'$ and $\langle T_1', S_1' \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F' \curlywedge \bar{F},low} \langle T_2', S_2' \rangle$. Notice that we have $\langle T_1', S_1' \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2' \rangle$.

In all three cases, we use Clause 7 and Subject Reduction (Theorem Appendix A.10) to conclude.

$\bar{M}_1$ is a value. In this case $P_1' = \{\bar{M}_1^m\}$, $F' = \mho$ and $\langle T_1', S_1' \rangle = \langle T_1, S_1 \rangle$. Then $\bar{M}_2 \in \boldsymbol{Val}$ by Remark Appendix A.18, so $W \vdash \langle \{{M_2}^m\}, T_2, S_2 \rangle \xrightarrow[F']{d} \langle \{\bar{M}_2^m\}, T_2, S_2 \rangle$. We conclude using Clause 1 and Subject Reduction (Theorem Appendix A.10).

□

We have seen in Remark Appendix A.18 that when two expressions are related by $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ and one of them is a value, then the other one is also a value. From a semantical point of view, when an expression has reached a value it means that it has successfully completed its computation. We will now see that when two expressions are related by $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ and one of them is unable to *resolve* into a value, in any sequence of unrelated computation steps, then the other one is also unable to do so. We shall use the notion of *derivative of an expression* M:

Definition Appendix A.21 (Derivative of an Expression)**.** *Given a fixed flow policy mapping W, we say that an expression M' is a* derivative *of an expression M if and only if*

- $M' = M$, *or*
- *there exist two states $\langle T_1, S_1\rangle$ and $\langle T'_1, S'_1\rangle$ and a derivative M'' of M such that, for some m, F, d, P:*

$$W \vdash \langle\{M''^m\}, T_1, S_1\rangle \xrightarrow[F]{d} \langle\{M'^m\} \cup P, T'_1, S'_1\rangle$$

Definition Appendix A.22 (Non-resolvable Expressions)**.** *Given a fixed flow policy mapping W, we say that an expression M is* non-resolvable*, denoted $M\dagger$, if there is no derivative M' of M such that $M' \in \boldsymbol{Val}$.*

Lemma Appendix A.23. *If for some F, low and j we have that $M\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ N$ and $M\dagger$, then $N\dagger$.*

Proof. Let us suppose that $\neg N\dagger$. That means that there exists a finite number of states $\langle T_1, S_1\rangle$, ..., $\langle T_n, S_n\rangle$, and $\langle T'_1, S'_1\rangle$, ..., $\langle T'_n, S'_n\rangle$, of expressions N_1, ..., N_n and of thread names $m_1, \ldots m_n$ such that

$$\begin{array}{rcl} W \vdash \langle\{N^{m_1}\}, T_1, S_1\rangle & \to & \langle\{N_1^{m_1}\}, T'_1, S'_1\rangle \text{ and} \\ W \vdash \langle\{N_1^{m_2}\}, T_2, S_2\rangle & \to & \langle\{N_2^{m_2}\}, T'_2, S'_2\rangle \text{ and} \\ & \vdots & \\ W \vdash \langle\{N_n^{m_n}\}, T_n, S_n\rangle & \to & \langle\{N_n^{m_n}\}, T'_n, S'_n\rangle \end{array}$$

and such that $N_n \in \boldsymbol{Val}$. By Strong Bisimulation for Low-Terminating Threads (Proposition Appendix A.20), we have that there exists a finite number of states $\langle \bar{T}'_1, \bar{S}'_1\rangle$, ..., $\langle \bar{T}'_n, \bar{S}'_n\rangle$, of expressions $\bar{M}_1$, ..., $\bar{M}_n$, of pools of threads $P_1, \ldots P_n$ such that

$$\begin{array}{rcl} W \vdash \langle\{M^{m_1}\}, T_1, S_1\rangle & \to & \langle\{M_1^{m_1}\} \cup P_1, \bar{T}'_1, \bar{S}'_1\rangle \text{ and} \\ W \vdash \langle\{M_1^{m_2}\}, T_2, S_2\rangle & \to & \langle\{M_2^{m_2}\} \cup P_2, \bar{T}'_2, \bar{S}'_2\rangle \text{ and} \\ & \vdots & \\ W \vdash \langle\{M_{n-1}^{m_n}\}, T_n, S_n\rangle & \to & \langle\{M_n^{m_n}\} \cup P_n, \bar{T}'_n, \bar{S}'_n\rangle \end{array}$$

such that:

$$M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{N}_1, \ \textit{and} \ \ldots, \ \textit{and} \ M_n \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ \bar{N}_n$$

By Remark Appendix A.18, we then have that $M_n \in \boldsymbol{Val}$. Since M_n is a derivative of M, we conclude that $\neg M\dagger$. □

The following lemma deduces operational "highness" of threads from that of its subexpressions.

Lemma Appendix A.24 (Composition of High Expressions)**.** *Suppose that M is typable with respect to $\Sigma.\Gamma, j$ and F. Then:*

1. *If $M = (M_1 \ M_2)$ and either*
   - *$M_1\dagger$ and ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,l}$, or*
   - *${M_1}^m, {M_2}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,l}$ and M_1 is a syntactically $(\Sigma, \Gamma, j, F, l)$-high function,*

   then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,l}$.
2. *If $M = (\text{if } M_1 \text{ then } M_t \text{ else } M_f)$ and ${M_1}^m, {M_t}^m, {M_f}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.*
3. *If $M = (\text{ref}_{l,\theta} \ M_1)$ and $l \not\sqsubseteq^F low$ and ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.*
4. *If $M = (M_1; M_2)$ and either*
   - *$M_1\dagger$ and ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, or*
   - *${M_1}^m, {M_2}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$,*

   then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.
5. *If $M = (M_1 := M_2)$ and M_1 has type $\theta \ \text{ref}_{l,n_k}$ with $l \not\sqsubseteq^F low$ and either*
   - *$M_1\dagger$ and ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, or*
   - *${M_1}^m, {M_2}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$,*

   then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.
6. *If $M = (\text{flow } F \text{ in } M_1)$ and ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.*
7. *If $M = (\text{allowed } F \text{ then } M_t \text{ else } M_f)$ and ${M_t}^m, {M_f}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, then $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.*

Lemma Appendix A.25. *If for some Σ, Γ, j, F, low and Υ we have that $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ and $M_1 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,l}$, then $M_2 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,l}$.*

Proof. By induction on the definition of $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$.

Clause 7. Here we have $M_1 = (\text{flow } F' \text{ in } \bar{M}_1)$ and $M_2 = (\text{flow } F' \text{ in } \bar{M}_2)$ with $\bar{M}_1 \ \mathcal{T}^{j}_{F \curlywedge F',low} \ \bar{M}_2$. Clearly we have that $\bar{M}_1 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, so by induction hypothesis also $\bar{M}_2 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Therefore, by Composition of High Expressions (Lemma Appendix A.24) we have that $M_2 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

□

We now present the main steps for proving soundness of the type system of Figure 4 with respect to the notion of security of Definition 3.3.

Behavior of Typable Low Expressions. In this second phase of the proof, we consider the general case of threads that are typable with any termination level. As in the previous subsection, we show that a typable expression behaves as a strong bisimulation, provided that it is operationally low. For this purpose, we make use of the properties identified for the class of low-terminating expressions by allowing only these to be followed by low-writes. Conversely, high-terminating expressions can only be followed by high-expressions (see Definitions Appendix A.13 and Appendix A.14).

Lemma Appendix A.26 (High Threads might Split)**.** *Consider a thread M^m for which there exist Γ, F, s and τ such that $\Gamma \vdash^{\Sigma}_{j,F} M : s, \tau$ and suppose that $M = \mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$ with $j \not\sqsubseteq^F low$. Then, we have that $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.*

Proof. By induction on the structure of E, using Lemma Appendix A.12 and Lemma Appendix A.15. Consider that we have $M = \mathrm{E}[M_0]$, where $M_0 = (\text{allowed } F' \text{ then } N_t \text{ else } N_f)$.

$\mathbf{E}[M_0] = M_0$. Then, by ALLOW, we have $\Gamma \vdash^{\Sigma}_{j,F} (\text{allowed } F' \text{ then } N_t \text{ else } N_f) : s, \tau$ where $\Gamma \vdash^{\Sigma}_{j,F} N_t : s_t, \tau$, $\Gamma \vdash^{\Sigma}_{j,F} N_f : s_f, \tau$ and $j \sqsubseteq^F s_t.w, s_f.w$. This means $s_t.w, s_f.w \not\sqsubseteq^F low$, so by Lemma Appendix A.15, then ${N_t}^m, {N_f}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. By Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (\mathbf{E_1}[M_0]\ M_1)$. Then by rule APP we have that $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1, \tau_1 \xrightarrow[F,j]{s_1'} \sigma_1$ and $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1'', \tau_1$ with $s_1.r \sqsubseteq^F s_1'.w$ and $s_1.r \sqsubseteq^F s_1''.w$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1.t$, which implies that $j \sqsubseteq^F s_1.t$ and $s_1.t \not\sqsubseteq^F low$. Therefore, $\mathrm{E}_1[M_0]$ is a syntactically (F, low, j)-high function and M_1 is (F, low, j)-high. By High Expressions (Lemma Appendix A.15) we have ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Lemma Appendix A.24, $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (V\ \mathbf{E_1}[M_0])$. Then by APP we have $\Gamma \vdash^{\Sigma}_{j,F} V : s_1, \tau_1 \xrightarrow[F,j]{s_1'} \sigma_1$ and $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1'', \tau_1$ with $s_1''.t \sqsubseteq^F s_1'.w$ and $s_1'.t \sqsubseteq^F s_1''.w$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1''.t$, which implies that $j \sqsubseteq^F s_1''.t$ and $s_1''.t \not\sqsubseteq^F low$, and so $s_1'.w \not\sqsubseteq^F low$. Therefore, $s_1'.w \not\sqsubseteq^F low$, and $s_1''.w \not\sqsubseteq^F low$, which means that V is a syntactically (F, low, j)-high function and $\mathrm{E}_1[M_0]$ is (F, low, j)-high. By induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (\mathbf{if}\ \mathbf{E_1}[M_0]\ \mathbf{then}\ M_t\ \mathbf{else}\ M_f)$. Then by rule COND we have that $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1, \mathsf{bool}$, and $\Gamma \vdash^{\Sigma}_{j,F} M_t : s_1', \tau_1$ and $\Gamma \vdash^{\Sigma}_{j,F} M_f : s_1'', \tau_1$ with $s_1.t \sqsubseteq^F s_1'.w, s_1'.w$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1.t$, which implies that $j \sqsubseteq^F s_1.t$ and $s_1.t \not\sqsubseteq^F low$. Therefore, $s_1'.w, s_1'.w \not\sqsubseteq^F low$, so by

High Expressions (Lemma Appendix A.15) we have ${M_t}^m, {M_t}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (\mathbf{E_1}[M_0]; M_1)$. Then by SEQ we have that $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1, \tau_1$ and $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1', \tau_1'$ with $s_1.t \sqsubseteq^F s_1'.w$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1.t$, which implies that $j \sqsubseteq^F s_1.t$ and $s_1.t \not\sqsubseteq^F \mathit{low}$. Therefore, $s_1'.w \not\sqsubseteq^F \mathit{low}$, and by High Expressions (Lemma Appendix A.15) we have ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (\mathbf{ref}_{l,\theta}\ \mathbf{E_1}[M_0])$. Then by REF we have that $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1, \theta$ with $s_1.t \sqsubseteq^F l$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1.t$, which implies that $j \sqsubseteq^F s_1.t$ and $s_1.t \not\sqsubseteq^F \mathit{low}$. Therefore, $l \not\sqsubseteq^F \mathit{low}$, and by induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (!\ \mathbf{E_1}[M_0])$. Easy, by induction hypothesis.

$\mathbf{E}[M_0] = (\mathbf{E_1}[M_0] := M_1)$. Then by ASS we have $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1, \theta\ \mathrm{ref}_{\bar{l}}$ and $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1', \tau_1$ with $s_1.t \sqsubseteq^F s_1'.w$ and $s_1.t \sqsubseteq^F \bar{l}$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1.t$, which implies that $j \sqsubseteq^F s_1.t$ and $s_1.t \not\sqsubseteq^F \mathit{low}$. Therefore, $\bar{l} \not\sqsubseteq^F \mathit{low}$ and $s_1'.w \not\sqsubseteq^F \mathit{low}$. Hence, by High Expressions (Lemma Appendix A.15) we have ${M_1}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (V := \mathbf{E_1}[M_0])$. Then by ASS we have $\Gamma \vdash^{\Sigma}_{j,F} V : s_1, \theta\ \mathrm{ref}_{\bar{l},n_{\bar{k}}}$ and $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}_1[M_0] : s_1', \tau_1$ with $s_1'.t \sqsubseteq^F \bar{l}$. By Lemma Appendix A.12 we have $j \sqsubseteq s_1'.t$, which implies that $j \sqsubseteq^F s_1'.t$ and $s_1'.t \not\sqsubseteq^F \mathit{low}$. Therefore, $\bar{l} \not\sqsubseteq^F \mathit{low}$, and by induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

$\mathbf{E}[M_0] = (\mathbf{flow}\ F'\ \mathbf{in}\ \mathbf{E_1}[M_0])$. Then by FLOW we have $\Gamma \vdash^{\Sigma}_{j,F \curlywedge F'} \mathrm{E}_1[M_0] : s_1, \tau_1$. By induction hypothesis $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F \curlywedge F',low}$, which implies $\mathrm{E}_1[M_0]^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Then, by Composition of High Expressions (Lemma Appendix A.24), we conclude that $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$.

□

We now design a binary relation on expressions that uses $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ to ensure that high-terminating expressions are always followed by operationally high ones. The definition of $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$ is given in Figure A.9. Notice that it is a symmetric relation. In order to ensure that expressions that are related by $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$

Definition Appendix A.27 ($\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$)**.** *We have that* $M_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ M_2$ *if* $\Gamma \vdash^{\Sigma}_{j,F} M_1 : s_1, \tau$ *and* $\Gamma \vdash^{\Sigma}_{j,F} M_2 : s_2, \tau$ *for some* Γ, s_1, s_2 *and* τ *and one of the following holds:*

Clause 1'. $M_1{}^m, M_2{}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, *or*

Clause 2'. $M_1 = M_2$, *or*

Clause 3'. $M_1 = (\text{if } \bar{M}_1 \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$ *and* $M_2 = (\text{if } \bar{M}_2 \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$ *with* $\bar{M}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $\bar{N_t}{}^m, \bar{M_f}{}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, *or*

Clause 4'. $M_1 = (\bar{M}_1\ \bar{N}_1)$ *and* $M_2 = (\bar{M}_2\ \bar{N}_2)$ *with* $\bar{M}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $\bar{N}_1^m, \bar{N}_2^m \in \mathcal{H}_{F,low}$, *and* $\bar{M}_1, \bar{M}_2$ *are syntactically* (F, low, j)*-high functions, or*

Clause 5'. $M_1 = (\bar{M}_1\ \bar{N}_1)$ *and* $M_2 = (\bar{M}_2\ \bar{N}_2)$ *with* $\bar{M}_1\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $\bar{N}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{N}_2$, *and* $\bar{M}_1, \bar{M}_2$ *are syntactically* (F, low, j)*-high functions, or*

Clause 6'. $M_1 = (\bar{M}_1; \bar{N})$ *and* $M_2 = (\bar{M}_2; \bar{N})$ *with* $\bar{M}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $\bar{N}^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, *or*

Clause 7'. $M_1 = (\bar{M}_1; \bar{N})$ *and* $M_2 = (\bar{M}_2; \bar{N})$ *with* $\bar{M}_1\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *or*

Clause 8'. $M_1 = (\text{ref}_{l,\theta}\ \bar{M}_1)$ *and* $M_2 = (\text{ref}_{l,\theta}\ \bar{M}_2)$ *with* $\bar{M}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $l \not\sqsubseteq^F low$, *or*

Clause 9'. $M_1 = (!\ \bar{M}_1)$ *and* $M_2 = (!\ \bar{M}_2)$ *with* $\bar{M}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *or*

Clause 10'. $M_1 = (\bar{M}_1 := \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 := \bar{N}_2)$ *with* $\bar{M}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $\bar{N}_1^m, \bar{N}_2^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, *and* $\bar{M}_1, \bar{M}_2$ *both have type* $\theta\ \text{ref}_{l,n_k}$ *for some* θ *and* l *such that* $l \not\sqsubseteq^F low$, *or*

Clause 11'. $M_1 = (\bar{M}_1 := \bar{N}_1)$ *and* $M_2 = (\bar{M}_2 := \bar{N}_2)$ *with* $\bar{M}_1\ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low}\ \bar{M}_2$, *and* $\bar{N}_1\ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low}\ \bar{N}_2$, *and* $\bar{M}_1, \bar{M}_2$ *both have type* $\theta\ \text{ref}_{l,n_k}$ *for some* θ *and* l *such that* $l \not\sqsubseteq^F low$, *or*

Clause 12'. $M_1 = (\text{flow } F' \text{ in } \bar{M}_1)$ *and* $M_2 = (\text{flow } F' \text{ in } \bar{M}_2)$ *with* $\bar{M}_1\ \mathcal{R}^{j}_{F \curlywedge F', low}\ \bar{M}_2$.

Figure A.9: The relation $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$

perform the same changes to the low memory, its definition requires that the references that are created or written using (potentially) different values are high, and that the body of the functions that are applied are syntactically high.

Remark Appendix A.28. *If* $M_1 \ \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \ M_2$*, then* $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$*.*

The above remark is used to prove the following lemma.

Lemma Appendix A.29. *If for some* j*,* F *and low we have that* $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ *and* $M_1 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$*, then* $M_2 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$*.*

Proof. By induction on the definition of $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$, using Lemma Appendix A.25.

□

We have seen in Splitting Computations (Lemma Appendix A.11) that two computations of the same expression can split only if the expression is about to read a reference that is given different values by the memories in which they compute. In Lemma Appendix A.30 we saw that the relation $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ relates the possible outcomes of expressions that are typable with a low termination effect. Finally, from the following lemma one can conclude that the above relation $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$ relates the possible outcomes of typable expressions in general.

Lemma Appendix A.30. *If for some* Σ*,* Γ*,* j *and* F *there exist* s*,* τ *such that* $\Gamma \vdash^{\Sigma}_{j,F} \mathrm{E}[(!\ a)] : s, \tau$ *with* $l \not\sqsubseteq_{F \curlywedge \lceil \mathrm{E} \rceil}$ *low, then for any values* $V_0, V_1 \in \boldsymbol{Val}$ *such that* $\Gamma \vdash^{\Sigma}_{,} V_i : \theta$ *we have* $\mathrm{E}[V_0] \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ \mathrm{E}[V_1]$*.*

Proof. By induction on the structure of E using Lemma Appendix A.9, Lemma Appendix A.19, Lemma Appendix A.15.

$\mathrm{E}[(!\ a)] = (\mathbf{flow}\ F'\ \mathbf{in}\ \mathrm{E}_1[(!\ a)])$. By rule FLOW we have $\Gamma \vdash^{\Sigma}_{j,F \curlywedge F'} V : s, \tau$. By induction hypothesis $\mathrm{E}_1[V_0] \ \mathcal{T}^{j}_{F \curlywedge F', low} \ \mathrm{E}_1[V_1]$, so we conclude by Lemma Appendix A.9 and Clause 12'.

□

We now state a crucial result of the paper: the relation $\mathcal{T}^{\Sigma,\Gamma}_{j,F,low}$ is a sort of "strong bisimulation".

Proposition Appendix A.31 (Strong Bisimulation for Typable Low Threads)**.**

If $M_1 \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2$ *and* $M_1 \notin \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$ *and for a given allowed flow policy mapping* W*, also* $W \vdash^{\Sigma,\Upsilon} \langle \{{M_1}^m\}, T_1, S_1 \rangle \xrightarrow[F']{d} \langle P_1', T_1', S_1' \rangle$*, with* $\Upsilon(m) = j$ *and* $\langle T_1, S_1 \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2, S_2 \rangle$ *then:*

- *If* $P_1' = \{{M_1'}^m\}$*, and* $a \in \mathrm{dom}(S_1' - S_1)$ *implies that* a *is fresh for* S_2*, then there exist* M_2'*,* T_2' *and* S_2' *such that* $W \vdash \langle \{{M_2}^m\}, T_2, S_2 \rangle \xrightarrow[F']{d} \langle \{{M_2'}^m\}, T_2', S_2' \rangle$ *with* $M_1' \ \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \ M_2'$ *and* $\langle T_1', S_1' \rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F', low} \langle T_2', S_2' \rangle$*.*

- *If* $P_1' = \{M_1'^m, N^n\}$ *for some expression* N *and thread name* $n \notin \mathrm{dom}(T_2)$, *then there exist* M_2', T_2' *and* S_2' *such that* $W \vdash \langle \{{M_2}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{{M_2'}^m, N^n\}, T_2', S_2'\rangle$ *with* $M_1' \; \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$ *and* $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$.

Proof. By case analysis on the clause by which $M_1 \; \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \; M_2$, and by induction on the definition of $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$. In the following, we use Subject Reduction (Theorem Appendix A.10) to guarantee typability (with the same type) for j, *low* and F, which is a requirement for being in the $\mathcal{R}^{\Sigma,\Gamma}_{j,F,low}$ relation. We also use the Strong Bisimulation for Low Terminating Threads Lemma (Lemma Appendix A.20).

Clause 2'. Here $M_1 = M_2$. By Guaranteed Transitions (Lemma Appendix A.16), then:

- If $P_1' = \{M_1'^m\}$, and $a \in \mathrm{dom}(S_1' - S_1)$ implies that a is fresh for S_2, then there exist M_2', T_2' and S_2' such that $W \vdash^{\Sigma,\Upsilon} \langle \{M_2^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{M_2'^m\}, T_2', S_2'\rangle$ with $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$. There are two cases to consider:

  $\boldsymbol{M_2' = M_1'}$**.** Then we have $M_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$, by Clause 2' and Subject Reduction (Theorem Appendix A.10).

  $\boldsymbol{M_2' \neq M_1'}$**.** Then by Splitting Computations (Lemma Appendix A.11) we have two possibilities:
  (1) there exists E and a such that $M_1' = \mathrm{E}[S_1(a)]$, $F' = \lceil \mathrm{E} \rceil$, $M_2' = \mathrm{E}[S_2(a)]$, $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$ and $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$. Since $S_1(a) \neq S_2(a)$, we have $\Sigma_1(a) \not\sqsubseteq^{F \curlywedge F'} low$. Therefore, $M_1' \; \mathcal{R}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$, by Lemma Appendix A.30 above.
  (2) there exists E such that $M_1' = \mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$, $F' = \lceil \mathrm{E} \rceil$, and $T_1(m) \neq T_2(m)$ with $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$ and $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$. Since $T_1(m) \neq T_2(m)$, we have $\Upsilon(m) = j \not\sqsubseteq^F low$, and by Lemma Appendix A.26 $M_1 \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$, which contradicts our assumption.

- If $P_1' = \{M_1'^m, N^n\}$, for some expression N and thread name $n \notin \mathrm{dom}(T_2)$ then there exist M_2', T_2' and S_2' such that we have $W \vdash^{\Sigma,\Upsilon} \langle \{M_2^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{M_2'^m, N^n\}, T_2', S_2'\rangle$ and with $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge F',low} \langle T_2', S_2'\rangle$ where $M_1' = M_2'$. Then we have $M_1' \; \mathcal{T}^{\Sigma,\Gamma}_{j,F,low} \; M_2'$, by Clause 2' and Subject Reduction (Theorem Appendix A.10).

Clause 12'. Here $M_1 = (\text{flow } F' \text{ in } \bar{M}_1)$ and $M_2 = (\text{flow } F' \text{ in } \bar{M}_2)$ with $\bar{M}_1 \; \mathcal{R}^{j}_{Fklmeet F',low} \; \bar{M}_2$. We can assume that $\bar{M}_1^m \notin \mathcal{H}^{\Sigma,\Upsilon}_{F \curlywedge F',low}$, since otherwise $\bar{M}_1^m \notin \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$ and by Composition of High Expressions (Lemma Appendix A.24) $M_1^m \in \mathcal{H}^{\Sigma,\Upsilon}_{F,low}$. Therefore $W \vdash^{\Sigma,\Upsilon} \langle \{\bar{M}_1^m\}, T_1, S_1\rangle \xrightarrow[F'']{d} \langle \{\bar{M}_1'^m\}, T_1', S_1'\rangle$ with $F' = \bar{F} \curlywedge F''$. By induction hypothesis, we have that

$W \vdash^{\Sigma,\Upsilon} \langle\{\bar{M}_2^m\}, T_2, S_2\rangle \xrightarrow[F'']{d} \langle\{\bar{M}_2'^m\}, T_2', S_2'\rangle$, and that $M_1' \; \mathcal{R}^j_{F \curlywedge \bar{F}, low} \; M_2'$ and also $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F \curlywedge \bar{F}, low} \langle T_2', S_2'\rangle$. Notice that $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{F, low} \langle T_2', S_2'\rangle$. We use Subject Reduction (Theorem Appendix A.10) and Clause 12' to conclude.

□

Behavior of Sets of Typable Threads. To conclude the proof of the Soundness Theorem, it remains to exhibit an appropriate bisimulation on thread configurations.

Definition Appendix A.32 ($\mathcal{R}^\Upsilon_{low}$)**.** *The relation $\mathcal{R}^\Upsilon_{low}$ is inductively defined as follows:*

$$a)\ \frac{M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}}{\{M^m\}\ \mathcal{R}^\Upsilon_{low}\ \emptyset} \qquad b)\ \frac{M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}}{\emptyset\ \mathcal{R}^\Upsilon_{low}\ \{M^m\}}$$

$$c)\ \frac{M_1\ \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),\mho,low}\ M_2}{\{{M_1}^m\}\ \mathcal{R}^\Upsilon_{low}\ \{{M_2}^m\}} \qquad d)\ \frac{P_1\ \mathcal{R}^\Upsilon_{low}\ P_2 \quad Q_1\ \mathcal{R}^\Upsilon_{low}\ Q_2}{P_1 \cup Q_1\ \mathcal{R}^\Upsilon_{low}\ P_2 \cup Q_2}$$

Proposition Appendix A.33. *The relation*

$$\mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} = \{(\langle P_1, T_1\rangle, \langle P_2, T_2\rangle) \mid P_1\ \mathcal{R}^\Upsilon_{low}\ P_2 \text{ and } T_1 =^{\Sigma,\Upsilon}_{\mho,l} T_2\}$$

is a $(W, \Sigma, \Upsilon, \Gamma, l)$-bisimulation acording to Definition Appendix A.3.

Proof. It is easy to see, by induction on the definition of $\mathcal{R}^\Upsilon_{low}$, that the relation $\mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low}$ is symmetric. We show, by induction on the definition of $\mathcal{R}^\Upsilon_{low}$, that if $\langle P_1, T_1\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle P_2, T_2\rangle$ and for any given S_1, S_2 such that $\langle T_1, S_1\rangle =^{\Sigma_1,\Upsilon}_{F,low} \langle T_2, S_2\rangle$ we have $W \vdash^{\Sigma,\Upsilon} \langle P_1, T_1, S_1\rangle \xrightarrow[F]{d} \langle P_1', T_1', S_1'\rangle$, and $(\mathrm{dom}(S_1') - \mathrm{dom}(S_1)) \cap \mathrm{dom}(S_2) = \emptyset$ and $(\mathrm{dom}(T_1') - \mathrm{dom}(T_1)) \cap \mathrm{dom}(T_2) = \emptyset$, then there exist T_2', P_2' and S_2' such that $W \vdash^{\Sigma,\Upsilon} \langle P_2, T_2, S_2\rangle \twoheadrightarrow \langle P_2', T_2', S_2'\rangle$ and $\langle P_1', T_1'\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle P_2', T_2'\rangle$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{\mho,low} \langle T_2', S_2'\rangle$.

Rule *a*). Then $P_1 = \{M^m\}$, $P_2 = \emptyset$, and $M^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}$. Therefore, $P_1' \subseteq \mathcal{H}^{\Sigma,\Upsilon}_{G,low}$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{\mho,low} \langle T_1, S_1\rangle$. We have that $W \vdash^{\Sigma,\Upsilon} \langle P_2, T_2, S_2\rangle \twoheadrightarrow \langle P_2, T_2, S_2\rangle$ and by transitivity $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{\mho,low} \langle T_2, S_2\rangle$. By Rules a) and d), we have $P_1' \; \mathcal{R}^\Upsilon_{low} \; \emptyset$. Then, $\langle P_1', T_1'\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle \emptyset, T_2'\rangle$.

Rule *c*). Then $P_1 = \{{M_1}^m\}$ and $P_2 = \{{M_2}^m\}$, and we have $M_1 \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),\mho,low} \; M_2$. If $M_1^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}$, then by Rule a), we have that $P_1' \; \mathcal{R}^\Upsilon_{low} \; \emptyset$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{\mho,low} \langle T_2', S_2'\rangle$. Also, by Lemma Appendix A.29, we have that $M_2^m \in \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}$, so by Rule b) $\emptyset \; \mathcal{R}^\Upsilon_{low} \; P_2$. By Rule d), we have $P_1' \; \mathcal{R}^\Upsilon_{low} \; P_2$. Then, $\langle P_1', T_1'\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle P_2', T_2'\rangle$.

If $M_1^m \notin \mathcal{H}^{\Sigma,\Upsilon}_{\mho,low}$, there are two cases to be considered:

$\boldsymbol{P_1' = \{M_1'^m\}}$**.** Then by Strong Bisimulation for Typable Low Threads (Proposition Appendix A.31) there exist T_2', M_2' and S_2' such that $W \vdash^{\Sigma,\Upsilon} \langle \{M_2{}^m\}, T_2, S_2\rangle \xrightarrow[F']{d} \langle \{M_2'^m\}, T_2', S_2'\rangle$ with $M_1' \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),\mathfrak{V},low} \; M_2'$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{\mathfrak{V},low} \langle T_2', S_2'\rangle$. Then, by Rule c), we have that $\{M_1'^m\} \; \mathcal{R}^{\Upsilon}_{low} \; \{M_2'^m\}$. Then, $\langle \{M_1'^m\}, T_1'\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle \{M_2'^m\}, T_2'\rangle$.

$\boldsymbol{P_1' = \{M_1'^m, N^n\}}$**.** We proceed as in the previous case to conclude that there exists $M_2'^m$ such that $\{M_1'^m\} \; \mathcal{R}^{\Upsilon}_{low} \; \{M_2'^m\}$. By Subject Reduction (Theorem Appendix A.10), by Lemma Appendix A.7, and by Clause 2' we have $N \; \mathcal{R}^{n}_{\mathfrak{V},low} \; N$, and so by Rule c) we have $\{N^n\} \; \mathcal{R}^{\Upsilon}_{low} \; \{N^n\}$. Therefore, by Rule d), we have $\{M_1'^m, N^n\} \; \mathcal{R}^{\Upsilon}_{low} \; \{M_2'^m, N^n\}$. Then, $\langle \{M_1'^m, N^n\}, T_1'\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle \{M_2'^m, N^n\}, T_2'\rangle$.

Rule *d*). Then $P_1 = \bar{P}_1 \cup \bar{Q}_1$ and $P_2 = \bar{P}_2 \cup \bar{Q}_2$, with $\bar{P}_1 \; \mathcal{R}^{\Upsilon}_{low} \; \bar{P}_2$ and $\bar{Q}_1 \; \mathcal{R}^{\Upsilon}_{low} \; \bar{Q}_2$. Suppose that $W \vdash^{\Sigma,\Upsilon} \langle \bar{P}_1, T_1, S_1\rangle \xrightarrow[F]{d} \langle \bar{P}_1', T_1', S_1'\rangle$ – the case where $\bar{Q}_1$ reduces is analogous. By induction hypothesis, there exist T_2', $\bar{P}_2'$ and S_2' such that $W \vdash^{\Sigma,\Upsilon} \langle \bar{P}_2, T_2, S_2\rangle \twoheadrightarrow \langle \bar{P}_2', T_2', S_2'\rangle$ with $\bar{P}_1' \; \mathcal{R}^{\Upsilon}_{low} \; \bar{P}_2'$ and $\langle T_1', S_1'\rangle =^{\Sigma_1,\Upsilon}_{\mathfrak{V},low} \langle T_2', S_2'\rangle$. Then, we have $W \vdash^{\Sigma,\Upsilon} \langle \bar{P}_2 \cup \bar{Q}_2, T_2, S_2\rangle \twoheadrightarrow \langle \bar{P}_2' \cup \bar{Q}_2, T_2', S_2'\rangle$, and by Rule d) we have $\bar{P}_1' \cup \bar{Q}_1 \; \mathcal{R}^{\Upsilon}_{low} \; \bar{P}_2' \cup \bar{Q}_2$. Then, $\langle \bar{P}_1' \cup \bar{Q}_1, T_1'\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle \bar{P}_2' \cup \bar{Q}_2, T_2'\rangle$.

□

We can now prove the main-result regarding Non-disclosure for Networks:

Theorem Appendix A.34 (Soundness of Typing Non-disclosure for Networks.)**.**
Consider a pool of threads P, an allowed-policy mapping W, a reference labeling Σ, a thread labeling Υ and a typing environment Γ. If for all $M^m \in P$ there exist s, and τ such that $\Gamma \vdash^{\Sigma}_{\Upsilon(m),\mathfrak{V}} M : s, \tau$, then P satisfies the Non-disclosure for Networks policy, i.e. $P \in \mathcal{NDN}_2(W, \Sigma, \Upsilon, \Gamma)$.

Proof. For all $M^m \in P$ and for all choices of security levels *low*, by assumption and by Clause 2' of Definition Appendix A.27, we have that $M \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; M$. By Rule c) of Definition Appendix A.32 we then have $\{M^m\} \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \{M^m\}$. Therefore, by Rule d) we have that $P \; \mathcal{R}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; P$, from which we conclude that for all position trackers T_1, T_2 such that $\mathrm{dom}(P) = \mathrm{dom}(T_1) = \mathrm{dom}(T_2)$ and $T_1 =^{\Sigma,\Upsilon}_{\mathfrak{V},l} T_2$ we have $\langle P, T_1\rangle \; \mathcal{B}^{\Sigma,\Gamma}_{\Upsilon(m),low} \; \langle P, T_2\rangle$. By Proposition Appendix A.33 we conclude that $\langle P, T_1\rangle \approx^{\Sigma,\Upsilon}_{\Gamma,low} \langle P, T_2\rangle$. □

Our soundness result for non-disclosure is compositional, in the sense that it is enough to verify the typability of each thread separately in order to ensure non-disclosure for the whole network.

Appendix B. Proofs for "Controlling Declassification"

Appendix B.1. Formalization of Flow Policy Confinement

In [2, 5] Flow Policy Confinement is defined for Networks, considering a distributed setting with code mobility, by means of a bisimulation on "located threads". In this paper we used a bisimulation on thread configurations, for the sake of uniformity with the definition on Non-disclosure for Networks, and for clarity. We prove that the two variations of the definition are equivalent.

Definition on located threads. The property is defined co-inductively on sets of located threads, consisting of pairs $\langle d, M^m\rangle$ that carry information about the location d of a thread M^m. The location of each thread determines which allowed flow policy it should obey at that point, and is used to place a restriction on the flow policies that decorate the transitions: at any step, they should comply to the allowed flow policy of the domain where the thread who performed it is located.

Definition Appendix B.1 ((W, Σ, Γ)-Confined Located Threads)**.** *Consider an allowed-policy mapping W, a reference labeling Σ, and a typing environment Γ. A set $\mathcal{CLT}$ of located threads is a set of (W, Σ, Γ)-*confined located threads *if the following holds for all $\langle d, M^m\rangle \in \mathcal{CLT}$, for all T such that $T(m) = d$, and for all (W, Σ, Γ)-compatible memory S:*

- $W \vdash^{\Sigma,\Upsilon} \langle\{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{M'^m\}, T', S'\rangle$ *implies* $W(T(m)) \preccurlyeq F$ *and also* $\langle T'(m), M'^m\rangle \in \mathcal{CLT}$.
- $W \vdash^{\Sigma,\Upsilon} \langle\{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{M'^m, N^n\}, T', S'\rangle$ *implies* $W(T(m)) \preccurlyeq F$ *and also* $\langle T'(m), M'^m\rangle, \langle T'(n), N^n\rangle \in \mathcal{CLT}$.

Furthermore, S' is still (W, Σ, Γ)-compatible. The largest set of (W, Σ, Γ)-confined located threads is denoted $\mathcal{CLT}_W^{\Sigma,\Gamma}$.

For any W, Σ and Γ, the set of located threads where threads are values is a set of (W, Σ, Γ)-confined located threads. Furthermore, the union of a family of sets of (W, Σ, Γ)-confined located threads is a set of (W, Σ, Γ)-confined located threads. Consequently, $\mathcal{CTC}_W^{\Sigma,\Gamma}$ exists.

We say that a thread M^m is (W, Σ, Γ)-confined when located at d, if $\langle d, M^m\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma}$. A well formed *thread configuration* $\langle P, T\rangle$, satisfying the applicable rules of a well formed configuration, is said to be (W, Σ, Γ)-confined if all located threads in $\{\langle T(m), M^m\rangle \mid M^m \in P\}$ are (W, Σ, Γ)-confined.

Definition Appendix B.2 (Flow Policy Confinement (on located threads))**.** *A pool of threads P satisfies Flow Policy Confinement with respect to an allowed-policy mapping W, a reference labeling Σ and a typing environment Γ, if for all domains $d \in \mathbf{Dom}$ and threads $M^m \in P$ we have that $\langle d, M^m\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma}$. We then write $P \in \mathcal{FPC}_1(W, \Sigma, \Gamma)$.*

Definition on thread configurations.

Definition Appendix B.3 ((W, Σ, Γ)-Confined Thread Configurations)**.** *Consider an allowed-policy mapping W, a reference labeling Σ, and a typing environment Γ. A set $\mathcal{CTC}$ of thread configurations is a set of (W, Σ, Γ)-*confined thread configurations *if it satisfies, for all P, T, and for all (W, Σ, Γ)-compatible memories S:*

$$\langle P, T\rangle \in \mathcal{CTC} \text{ and } W \vdash \langle P, T, S\rangle \xrightarrow[F]{d} \langle P', T', S'\rangle \text{ implies}$$
$$W(d) \preccurlyeq F \text{ and } \langle P', T'\rangle \in \mathcal{CTC}$$

Furthermore, S' is still (W, Σ, Γ)-compatible. The largest set of (W, Σ, Γ)-confined thread configurations is denoted $\mathcal{CTC}_W^{\Sigma,\Gamma}$.

For any W, Σ and Γ, the set of thread configurations where threads are values is a set of (W, Σ, Γ)-confined thread configurations. Furthermore, the union of a family of (W, Σ, Γ)-confined thread configurations is a (W, Σ, Γ)-confined thread configurations. Consequently, $\mathcal{CTC}_W^{\Sigma,\Gamma}$ exists.

Definition Appendix B.4 (Flow Policy Confinement (on thread configurations))**.** *A pool of threads P satisfies Flow Policy Confinement with respect to an allowed-policy mapping W, a reference labeling Σ and a typing environment Γ, if for all thread configurations $\langle P, T\rangle$ such that $\mathrm{dom}(P) = \mathrm{dom}(T)$ we have that $\langle P, T\rangle \in \mathcal{CTC}_W^{\Sigma,\Gamma}$. We then write $P \in \mathcal{FPC}_2(W, \Sigma, \Gamma)$.*

Note that for any $P_1 \subseteq P$ and $T_1 \subseteq T$ such that $\mathrm{dom}(P) = \mathrm{dom}(T)$, if $\langle P, T\rangle \in \mathcal{CTC}_W^{\Sigma,\Gamma}$ then $\langle P_1, T_1\rangle \in \mathcal{CTC}_W^{\Sigma,\Gamma}$.

Comparison. Flow Policy Confinement, defined over thread configurations, is equivalent to when defined over located threads.

Proposition Appendix B.5. $\mathcal{FPC}_1(W, \Sigma, \Gamma) = \mathcal{FPC}_2(W, \Sigma, \Gamma)$.

Proof. We assume that $P \in \mathcal{FPC}_1(W, \Sigma, \Gamma)$, and show that the set

$$\mathcal{CTC} = \{\langle P, T\rangle \mid \forall M^m \in P \,.\, \langle T(m), M^m\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma} \text{ and } \mathrm{dom}(P) = \mathrm{dom}(T)\}$$

satisfies $\mathcal{CTC} \subseteq \mathcal{CTC}_W^{\Sigma,\Gamma}$. Given any thread configuration $\langle P, T\rangle \in \mathcal{CTC}$, and any thread $M^m \in P$, it is clear that $\langle T(m), M^m\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma}$. If

$$W \vdash^{\Sigma,\Upsilon} \langle P, T, S\rangle \xrightarrow[F]{d} \langle P', T', S'\rangle$$

then for some P'', T'' such that $P = \{M^m\} \cup P''$ and $T = \{m \mapsto d\} \cup T''$ we have that either:

- $W \vdash^{\Sigma,\Upsilon} \langle \{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{M'^m\}, T', S'\rangle$ where $P' = \{M'^m\} \cup P''$ and $T' = \{m \mapsto d\} \cup T''$. In this case, by hypothesis, $W(T(m)) \preccurlyeq F$ and also $\langle T'(m), M'^m\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma}$.

- $W \vdash^{\Sigma,\Upsilon} \langle\{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{M'^m, N^n\}, T', S'\rangle$ where $P' = \{M'^m, N^n\} \cup P''$ and $T' = \{m \mapsto d, n \mapsto d'\} \cup T''$ for some d'. In this case, by hypothesis, $W(T(m)) \preccurlyeq F$ and also $\langle T'(m), M'^m\rangle, \langle T'(n), N^n\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma}$.

In both cases, we can conclude that $\forall M^m \in P' \,.\, \langle T'(m), M^m\rangle \in \mathcal{CLT}_W^{\Sigma,\Gamma}$ and $\mathrm{dom}(P') = \mathrm{dom}(T')$, so $\langle P', T'\rangle \in \mathcal{CTC}$. We then have that $\mathcal{CTC} \subseteq \mathcal{CTC}_W^{\Sigma,\Gamma}$, and also that for all $\langle P, T\rangle$ such that $\mathrm{dom}(P) = \mathrm{dom}(T)$, we have that $\langle P, T\rangle \in \mathcal{CTC}$. Then, $P \in \mathcal{FPC}_2(W, \Sigma, \Gamma)$.

We now assume that $P \in \mathcal{FPC}_2(W, \Sigma, \Gamma)$, and show that the set

$$\mathcal{CLT} = \{\langle T(m), M^m\rangle \mid \exists P, T \,.\, \langle P, T\rangle \in \mathcal{CTC}_W^{\Sigma,\Gamma} \text{ and } M^m \in P\}$$

satisfies $\mathcal{CLT} \subseteq \mathcal{CLT}_W^{\Sigma,\Gamma}$. Given any located thread $\langle d, M^m\rangle \in \mathcal{CLT}$ and position tracker T such that $T(m) = d$, and any (W, Σ, Γ)-compatible memory S, it is clear that also $\langle\{M^m\}, \{m \mapsto d\}\rangle \in \mathcal{CTC}_W^{\Sigma,\Gamma}$. Then, if:

- $W \vdash^{\Sigma,\Upsilon} \langle\{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{M'^m\}, T', S'\rangle$, then by hypothesis, $W(d) \preccurlyeq F$ and $\langle\{M'^m\}, T'\rangle \in \mathcal{FPC}_2(W, \Sigma, \Gamma)$. Therefore, $\langle T'(m), M'^m\rangle \in \mathcal{CLT}$.
- $W \vdash^{\Sigma,\Upsilon} \langle\{M^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{M'^m, N^n\}, T', S'\rangle$, then by hypothesis, $W(d) \preccurlyeq F$ and $\langle\{M'^m, N^n\}, T'\rangle \in \mathcal{FPC}_2(W, \Sigma, \Gamma)$. Therefore, both $\langle T'(m), M'^m\rangle$, $\langle T'(n), N'^n\rangle \in \mathcal{CLT}$.

We then have that $\mathcal{CLT} \subseteq \mathcal{CLT}_W^{\Sigma,\Gamma}$, and also that for all $\langle d, M^m\rangle$ such that $M^m \in P$, we have that $\langle d, M^m\rangle \in \mathcal{CLT}$. Then, $P \in \mathcal{FPC}_1(W, \Sigma, \Gamma)$. □

Appendix B.2. Type System

Appendix B.2.1. Subject reduction

In order to establish the soundness of the type system of Figure 5 we need a Subject Reduction result, stating that types that are given to expressions are preserved by computation. To prove it we follow the usual steps [41].

Remark Appendix B.6.

1. *If* $W \in \boldsymbol{Pse}$ *and* $W; \Gamma \vdash_A^\Sigma W : \tau$*, then for all flow policies* A'*, we have that* $W; \Gamma \vdash_{A'}^\Sigma W : \tau$*.*
2. *For any flow policies* A *and* A' *such that* $A' \preccurlyeq A$*, we have that* $W; \Gamma \vdash_A^\Sigma M : \tau$ *implies* $W; \Gamma \vdash_{A'}^\Sigma M : \tau$*.*

Lemma Appendix B.7.

1. *If* $W; \Gamma \vdash_A^\Sigma M : \tau$ *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $W; \Gamma, x : \sigma \vdash_A^\Sigma M : \tau$*.*
2. *If* $W; \Gamma, x : \sigma \vdash_A^\Sigma M : \tau$ *and* $x \notin \mathrm{fv}(M)$ *then* $W; \Gamma \vdash_A^\Sigma M : \tau$*.*

Proof. By induction on the inference of the type judgment. □

Lemma Appendix B.8 (Substitution)**.**
If $W; \Gamma, x : \sigma \vdash_A^\Sigma M : \tau$ *and* $W; \Gamma \vdash_{A'}^\Sigma W : \sigma$ *then* $W; \Gamma \vdash_A^\Sigma \{x \mapsto W\}M : \tau$*.*

Proof. By induction on the inference of $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} M : \tau$, and by case analysis on the last rule used in this typing proof, using the previous lemma.

Nil. Here $\{x \mapsto W\}M = M$, and since $x \notin \mathrm{fv}(M)$ then by Lemma Appendix B.7 we have $\Gamma \vdash^{\Sigma}_{j} M : s, \tau$.

Var. If $M = x$ then $\sigma = \tau$ and $\{x \mapsto W\}M = W$. By Remark Appendix B.6, we have $W;\Gamma \vdash^{\Sigma}_{A} W : \tau$. If $M \neq x$ then $\{x \mapsto W\}M = M$, where $x \notin \mathrm{fv}(M)$. Therefore, by Lemma Appendix B.7, we have $W;\Gamma \vdash^{\Sigma}_{A} M : \tau$.

Abs. Here $M = (\lambda y.\bar{M})$, and $W;\Gamma, x : \sigma, y : \bar{\tau} \vdash^{\Sigma}_{\bar{A}} \bar{M} : \bar{\sigma}$ where $\tau = \bar{\tau} \underset{\bar{A}}{\longrightarrow} \bar{\sigma}$. We can assume that $y \notin \mathrm{dom}(W;\Gamma, x : \sigma)$ (otherwise rename y). Therefore $\{x \mapsto W\}(\lambda y.\bar{M}) = (\lambda y.\{x \mapsto W\}\bar{M})$. By assumption and Lemma Appendix B.7 we can write $W;\Gamma, y : \bar{\tau} \vdash^{\Sigma}_{A'} W : \sigma$. By induction hypothesis, $W;\Gamma, y : \bar{\tau} \vdash^{\Sigma}_{\bar{A}} \{x \mapsto W\}\bar{M} : \bar{\sigma}$. Then, by Abs, $W;\Gamma \vdash^{\Sigma}_{A} (\lambda y.\{x \mapsto W\}\bar{M}) : \tau$.

Rec. Here $M = (\varrho y.\bar{W})$, and $W;\Gamma, x : \sigma, y : \tau \vdash^{\Sigma}_{A} \bar{W} : \tau$. We can assume that $y \notin \mathrm{dom}(W;\Gamma, x : \sigma)$ (otherwise rename y). Therefore $\{x \mapsto W\}(\varrho y.\bar{W}) = (\varrho y.\{x \mapsto W\}\bar{W})$. By assumption and Lemma Appendix B.7 we have $W;\Gamma, y : \tau \vdash^{\Sigma}_{A'} W : \sigma$. By induction hypothesis, $W;\Gamma, y : \tau \vdash^{\Sigma}_{A} \{x \mapsto W\}\bar{W} : \tau$. Then, by Rec, $W;\Gamma \vdash^{\Sigma}_{A} (\varrho y.\{x \mapsto W\}\bar{W}) : \tau$.

Cond. Here $M = (\text{if } \bar{M} \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$ and we have $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \bar{M} : \mathsf{bool}$, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} N_t : \tau_1$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} N_f : \tau_2$. By induction hypothesis, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \{x \mapsto W\}\bar{M} : \mathsf{bool}$, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \{x \mapsto W\}N_t : \tau_1$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \{x \mapsto W\}N_f : \tau_2$. Therefore, we have that $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A}$ $(\text{if } \{x \mapsto W\}\bar{M} \text{ then } \{x \mapsto W\}N_t \text{ else } \{x \mapsto W\}N_f) : \tau$ by rule Cond.

Mig. Here $M = (\text{thread}_l\ \bar{M} \text{ at } d)$ and we have that $W;\Gamma, x : \sigma \vdash^{\Sigma}_{W(d)} \bar{M} : \tau$, with $\tau = \mathsf{unit}$. By induction hypothesis, then $W;\Gamma, x : \sigma \vdash^{\Sigma}_{W(d)} \{x \mapsto W\}\bar{M} : \tau$. Therefore, by rule Mig, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} (\text{thread}_l\ \{x \mapsto W\}\bar{M} \text{ at } d) : \tau$.

Flow. Here $M = (\text{flow } \bar{F} \text{ in } \bar{M})$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \bar{M} : \tau$, with $A \preccurlyeq \bar{F}$. By induction hypothesis, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \{x \mapsto W\}\bar{M} : \tau$. Then, by Flow, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} (\text{flow } \bar{F} \text{ in } \{x \mapsto W\}\bar{M}) : \tau$.

Allow. Here $M = (\text{allowed } \bar{F} \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \bar{M} : \mathsf{bool}$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A \curlywedge \bar{F}} \bar{N}_t : \tau_1$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A} \bar{N}_f : \tau_2$. By induction hypothesis, $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A \curlywedge \bar{F}} \{x \mapsto W\}\bar{N}_t : \tau_1$ and $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A}$ $\{x \mapsto W\}\bar{N}_f : \tau_2$. Therefore, by rule Cond, we have that $W;\Gamma, x : \sigma \vdash^{\Sigma}_{A}$ $(\text{allowed } \bar{F} \text{ then } \{x \mapsto W\}N_t \text{ else } \{x \mapsto W\}N_f) : s, \tau$.

The proofs for the cases Loc, BT and BF are analogous to the one for Nil, while the proofs for Ref, App, Seq, Der and Ass are analogous to the one for Cond. □

Lemma Appendix B.9 (Replacement)**.**
If $W;\Gamma \vdash^{\Sigma}_{A} \mathrm{E}[M] : \tau$ *is a valid judgment, then the proof gives* M *a typing* $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \mathrm{E} \rceil} M : \bar{\tau}$ *for some* $\bar{\tau}$*. In this case, if* $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \mathrm{E} \rceil} N : \bar{\tau}$*, then* $W;\Gamma \vdash^{\Sigma}_{A} \mathrm{E}[N] : \tau$*.*

Proof. By induction on the structure of E.

$\mathrm{E}[M] = M$. This case is direct.

$\mathrm{E}[M] =$ (if $\bar{\mathrm{E}}[M]$ then $\bar{N}_t$ else $\bar{N}_f$). By Cond, we have $W;\Gamma \vdash^{\Sigma}_{A} \bar{\mathrm{E}}[M] :$ bool, and also $W;\Gamma \vdash^{\Sigma}_{A} \bar{N}_t : \tau$ and $W;\Gamma \vdash^{\Sigma}_{A} \bar{N}_f : \tau$. By induction hypothesis, the proof gives M a typing $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} M : \hat{\tau}$, for some $\hat{\tau}$.

Also by induction hypothesis, $W;\Gamma \vdash^{\Sigma}_{A} \bar{\mathrm{E}}[N] :$ bool. Again by Cond, we have $W;\Gamma \vdash^{\Sigma}_{A}$ (if $\bar{\mathrm{E}}[N]$ then $\bar{N}_t$ else $\bar{N}_f$) $: \tau$.

$\mathrm{E}[M] =$ (flow $\bar{F}$ in $\bar{\mathrm{E}}[M]$). By Flow, we have $W;\Gamma \vdash^{\Sigma}_{A} \bar{\mathrm{E}}[M] : \tau$ and $A \preccurlyeq \bar{F}$. By induction hypothesis, the proof gives M a typing $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} M : \hat{\tau}$, for some $\hat{\tau}$.

Also by induction hypothesis, $W;\Gamma \vdash^{\Sigma}_{A} \bar{\mathrm{E}}[N] : \tau$. Then, again by Flow, we have $W;\Gamma \vdash^{\Sigma}_{A}$ (flow $\bar{F}$ in $\bar{\mathrm{E}}[N]$) $: \tau$.

The proofs for the cases $\mathrm{E}[M] = \lceil (\mathrm{E}[M] := N) \rceil$, $\mathrm{E}[M] = \lceil (V := \mathrm{E}[M]) \rceil$, $\mathrm{E}[M] = \lceil (!\ \mathrm{E}[M]) \rceil$, $\mathrm{E}[M] = \lceil (\mathrm{E}[M]\ N) \rceil$, $\mathrm{E}[M] = \lceil (V\ \mathrm{E}[M]) \rceil$, $\mathrm{E}[M] = \lceil (\mathrm{E}[M]; N) \rceil$ and $\mathrm{E}[M] = \lceil \mathsf{ref}_{l,\theta} \mathrm{E}[M] \rceil$, are all analogous to the proof for the case $\mathrm{E}[M] =$ (if $\bar{\mathrm{E}}[M]$ then N_t else N_f). □

We check that the type of a thread and the compatibility of memories is preserved by reduction.

Proposition Appendix B.10 (Subject Reduction)**.** *Given a reference and thread labeling* Σ*,* Υ*, consider a thread* M^m *for which there exist* Γ*,* A *and* τ *such that* $W;\Gamma \vdash^{\Sigma}_{A} M : \tau$ *and suppose that* $W \vdash \langle \{M^m\}, T, S \rangle \xrightarrow[F]{d} \langle \{M'^m\} \cup P, T', S' \rangle$*, for a memory* S *that is* (W, Σ, Γ)*-compatible. Then,* $W;\Gamma \vdash^{\Sigma}_{A \curlywedge W(T(m))} M' : \tau$*, and* S' *is also* (W, Σ, Γ)*-compatible. Furthermore, if* $P = \{N^n\}$*, for some expression* N *and thread name* n*, then* $W;\Gamma \vdash^{\Sigma}_{W(T'(n))} N :$ unit*.*

Proof. Suppose that we have $M = \bar{\mathrm{E}}[\bar{M}]$ and that $W \vdash \langle \{\bar{M}^m\}, T, S \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{M}'^m\} \cup P', \bar{T}', \bar{S}' \rangle$. We start by observing that this implies $F = \bar{F} \curlywedge \lceil \bar{\mathrm{E}} \rceil$, $M' = \bar{\mathrm{E}}[\bar{M}']$, $P = P'$, $\bar{T}' = T'$ and $\bar{S}' = S'$. We can assume, without loss of generality, that $\bar{M}$ is the smallest in the sense that there is no $\hat{\mathrm{E}}, \hat{M}, \hat{N}$ such that $\hat{\mathrm{E}} \neq []$ and $\hat{\mathrm{E}}[\hat{M}] = \bar{M}$ for which we can write $W \vdash \langle \{\hat{M}^m\}, T, S \rangle \xrightarrow[\hat{F}]{d} \langle \{\hat{M}'^m\} \cup P, T', S' \rangle$.

By Lemma Appendix B.9, we have $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} \bar{M} : \bar{\tau}$ in the proof of $W;\Gamma \vdash^{\Sigma}_{A} \bar{\mathrm{E}}[\bar{M}] : \tau$, for some $\bar{\tau}$. We proceed by case analysis on the transition $W \vdash \langle \{\bar{M}^m\}, T, S \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{M}'^m\} \cup P, T', S' \rangle$, and prove that if $S' \neq S$ then:

- There exists $\bar{\tau}'$ such that $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} \bar{M}' : \bar{\tau}'$ and $\bar{\tau} \preccurlyeq \bar{\tau}'$. Furthermore, for every reference $a \in \mathrm{dom}(S')$ implies $\Gamma \vdash^{\Sigma}_{\mathcal{U}} S'(a) : \Sigma_2(a)$.
- If $P = \{N''^n\}$ for some expression N'' and thread name n, then also $W;\Gamma \vdash^{\Sigma}_{W(T'(n))} N : \mathsf{unit}$. (Note that in this case $S = S'$.)

By case analysis on the structure of $\bar{M}$:

$\bar{M} = ((\lambda x.\hat{M})\ V)$. Here we have $\bar{M}' = \{x \mapsto V\}\hat{M}$, $S = S'$ and $P = \emptyset$. By rule App, there exist $\hat{\tau}$ and $\hat{\sigma}$ such that $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} (\lambda x.\hat{M}) : \hat{\tau} \xrightarrow[A \curlywedge \lceil \bar{\mathrm{E}} \rceil]{} \hat{\sigma}$ and $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} V : \hat{\tau}$ with $\hat{\sigma} = \bar{\tau}$. By Abs, then $W;\Gamma, x : \hat{\tau} \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} \hat{M} : \hat{\sigma}$. Therefore, by Lemma Appendix B.8, we get $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} \{x \mapsto V\}\hat{M} : \bar{\tau}$. By Remark Appendix B.6, $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} \{x \mapsto V\}\hat{M} : \bar{\tau}$.

$\bar{M} = (\varrho x.W)$. Here we have $\bar{M}' = (\{x \mapsto (\varrho x.W)\}\ W)$, $S = S'$ and $P = \emptyset$. By rule Rec, we have $W;\Gamma, x : \bar{\tau} \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} W : \bar{\tau}$. Therefore, by Lemma Appendix B.8, we get $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} \{x \mapsto (\varrho x.W)\}W : \bar{\tau}$. By Remark Appendix B.6, $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} \{x \mapsto (\varrho x.W)\}W : \bar{\tau}$.

$\bar{M} = (\mathbf{if}\ tt\ \mathbf{then}\ N_t\ \mathbf{else}\ N_f)$. Here we have $\bar{M}' = N_t$, $S = S'$ and $P = \emptyset$. By Cond, we have that $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} N_t : \bar{\tau}$. By Remark Appendix B.6, $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} N_t : \bar{\tau}$.

$\bar{M} = (\mathbf{ref}_{l,\theta}\ V)$. Here we have $\bar{M}' = a$, $lab = a : \theta\ \mathrm{ref}_l$ for some reference name a, type θ and security level l, $S' = S \cup \{(a, V)\}$ and $P = \emptyset$. By Ref, $\bar{\tau} = \theta$ ref and $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} V : \theta$, and by Remark Appendix B.6 then $W;\Gamma \vdash^{\Sigma}_{\mathcal{U}} V : \theta$. By Lemma Appendix B.7 we have $W;\Gamma \vdash^{\Sigma}_{\mathcal{U}} S'(a) : \theta$ for every $a \in \mathrm{dom}(S')$. By Loc, we have $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} a : \theta$ ref, and $\bar{\tau} = \theta$ ref.

$\bar{M} = (!\ a)$. Here we have $\bar{M}' = S(a)$, $S = S'$ and $P = \emptyset$. By Der, we have that $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} a : \bar{\tau}$ ref, and by Loc we know that $\Sigma_2(a) = \bar{\tau}$. By compatibility assumption, then $W;\Gamma \vdash^{\Sigma}_{\mathcal{U}} S(a) : \Sigma_2(a)$, and by Remark Appendix B.6, then $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} S(a) : \bar{\tau}$.

$\bar{M} = (a := V)$. Here we have $\bar{M}' = ()$, and $P = \emptyset$. By Ass, $\bar{\tau} = \mathsf{unit}$, and $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} a : \theta$ ref and $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} V : \theta$, for some θ. By Loc, $\theta = \Sigma_2(a)$ and by Remark Appendix B.6 we have $W;\Gamma \vdash^{\Sigma}_{\mathcal{U}} V : \Sigma_2(a)$. By Nil, we have that $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} () : \bar{\tau}$, with $\bar{\tau} = \mathsf{unit}$.

$\bar{M} = (\mathbf{flow}\ F'\ \mathbf{in}\ V)$. Here we have $\bar{M}' = V$, $S = S'$ and $P = \emptyset$. By rule Flow, we have that $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} V : \bar{\tau}$ and by Remark Appendix B.6, we have $W;\Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} V : \bar{\tau}$.

$\bar{M} = (\textbf{allowed } F' \textbf{ then } N_t \textbf{ else } N_f) \textbf{ and } W(T(m)) \preccurlyeq F'$. Here we have $\bar{M}' = N_t$, $S = S'$ and $P = \emptyset$. By ALLOW, we have that $W; \Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge F'} N_t : \bar{\tau}$. By Remark Appendix B.6, $W; \Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} N_t : \bar{\tau}$.

$\bar{M} = (\textbf{allowed } F' \textbf{ then } N_t \textbf{ else } N_f) \textbf{ and } W(T(m)) \not\preccurlyeq F'$. Here we have $\bar{M}' = N_f$, $S = S'$ and $P = \emptyset$. By ALLOW, we have that $W; \Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil} N_f : \bar{\tau}$. By Remark Appendix B.6, $W; \Gamma \vdash^{\Sigma}_{A \curlywedge \lceil \bar{\mathrm{E}} \rceil \curlywedge W(T(m))} N_f : \bar{\tau}$.

$\bar{M} = (\textbf{thread}_k\ N \textbf{ at } d)$. Here we have $\bar{M}' = ()$, $P = \{\hat{N}^n\}$ for some thread name n, $S = S'$, and $T'(n) = d$. By MIG, since $A \curlywedge W(d) \curlywedge \lceil \bar{\mathrm{E}} \rceil \preccurlyeq W(d)$, we have that $W; \Gamma \vdash^{\Sigma}_{A \curlywedge W(d) \curlywedge \lceil \bar{\mathrm{E}} \rceil} N : \mathsf{unit}$ and $\bar{\tau} = \mathsf{unit}$, and by NIL we have that $W; \Gamma \vdash^{\Sigma}_{A} () : \mathsf{unit}$.

The cases $\bar{M} = (\text{if } ff \text{ then } N_t \text{ else } N_f)$ and $\bar{M} = (V; \hat{M})$ are analogous to the one for $\bar{M} = (\text{if } tt \text{ then } N_t \text{ else } N_f)$.

By Lemma Appendix B.9, we can finally conclude that: $W; \Gamma \vdash^{\Sigma}_{A \curlywedge W(T(m))} \bar{\mathrm{E}}[\bar{M}'] : \tau$. □

Appendix B.3. Runtime Type Checking

Appendix B.3.1. Subject Reduction

Proposition Appendix B.11 (Subject Reduction)**.** *Given a reference and thread labeling Σ, Υ, consider a thread M^m for which there exist Γ, A and τ such that $\Gamma \vdash^{\Sigma}_{A} M : \tau$ and suppose that $W \vdash \langle \{M^m\}, T, S \rangle \xrightarrow[F]{d} \langle \{M'^m\} \cup P, T', S' \rangle$, for a memory S that is (W, Σ, Γ)-compatible. Then, $\Gamma \vdash^{\Sigma}_{A \curlywedge W(T(m))} M' : \tau$, and S' is also (W, Σ, Γ)-compatible. Furthermore, if $P = \{N^n\}$, for some expression N and thread name n, then $\Gamma \vdash^{\Sigma}_{W(T'(n))} N : \mathsf{unit}$.*

Proof. The proof is the same as the one for Proposition Appendix B.10, with exception for the treatment of the case where the step corresponds to the creation of a new thread:

$\bar{M} = (\textbf{thread}_l\ N \textbf{ at } d)$. Here we have $\bar{M}' = ()$, $P = \{\hat{N}^n\}$ for some thread name n, $T'(n) = d$ and (by the migration condition) $\Gamma \vdash^{\Sigma}_{W(d)} N : \mathsf{unit}$, where $\bar{\tau} = \mathsf{unit}$. By NIL we have that $\Gamma \vdash^{\Sigma}_{A} () : \mathsf{unit}$.

□

Appendix B.4. Declassification effect

Appendix B.4.1. Subject Reduction

In order to prove Subject Reduction, we follow the usual steps [41].

Remark Appendix B.12. *If $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$ then $M \in \textbf{Val}$ iff $N \in \textbf{Val}$ and $s = \mho$.*

Remark Appendix B.13. *If* $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$ *and* $M \in \boldsymbol{Pse}$ *with* $\mathrm{rn}(M) \subseteq \mathrm{dom}(\Sigma)$*, then* $N \in \boldsymbol{Pse}$ *and* $s = \mho$*.*

Lemma Appendix B.14.

1. *If* $\Gamma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau$ *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $W; \Gamma, x : \sigma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau$*.*
2. *If* $W; \Gamma, x : \sigma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau$ *and* $x \notin \mathrm{fv}(M)$ *then* $\Gamma \vdash^{\Sigma} M \hookrightarrow \hat{M} : s, \tau$*.*

Proof. By induction on the inference of the type judgment. □

Lemma Appendix B.15 (Substitution)**.**
If $W; \Gamma, x : \sigma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$ *and* $\Gamma \vdash^{\Sigma} X_1 \hookrightarrow X_2 : \mho, \sigma'$ *with* $\sigma \preccurlyeq \sigma'$*, then* $\Gamma \vdash^{\Sigma} \{x \mapsto X_1\}M \hookrightarrow \{x \mapsto X_2\}N : s, \tau'$ *with* $\tau \preccurlyeq \tau'$*.*

Proof. By induction on the inference of $W; \Gamma, x : \sigma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$, and by case analysis on the last rule used in this typing proof, using Lemma Appendix B.14.

Nil$_\mathbf{I}$. Here $\{x \mapsto X_1\}M = M$, and $\{x \mapsto X_2\}N = N$, and since $x \notin \mathrm{fv}(M)$ then by Lemma Appendix B.14 we have $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$.

Var$_\mathbf{I}$. If $M = x$ then $N = x$, $s = \mho$, $\sigma = \tau$, $\{x \mapsto X_1\}M = X_1$, and $\{x \mapsto X_2\}N = X_2$. It is then direct that $\Gamma \vdash^{\Sigma} X_1 \hookrightarrow X_2 : s, \sigma'$, and we take $\tau' = \sigma'$. If $M \neq x$ then $N \neq x$, $\{x \mapsto X_1\}M = M$ and $\{x \mapsto X_1\}M = M$ where $x \notin \mathrm{fv}(M)$. Therefore, by Lemma Appendix B.14, we have $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$.

Abs$_\mathbf{I}$. Here $M = (\lambda y.\bar{M})$, $N = (\lambda y.\bar{N})$, $s = \mho$, and $W; \Gamma, x : \sigma, y : \bar{\tau} \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \bar{\sigma}$ where $\tau = \bar{\tau} \xrightarrow{\bar{s}} \bar{\sigma}$. We can assume that $y \notin \mathrm{dom}(W; \Gamma, x : \sigma)$ (otherwise rename y). Then $\{x \mapsto X_1\}(\lambda y.\bar{M}) = (\lambda y.\{x \mapsto X_1\}\bar{M})$ and $\{x \mapsto X_2\}(\lambda y.\bar{N}) = (\lambda y.\{x \mapsto X_2\}\bar{N})$. By assumption and Lemma Appendix B.14 we can write $W; \Gamma, y : \bar{\tau} \vdash^{\Sigma} X_1 \hookrightarrow X_2 : \mho, \sigma'$. By induction hypothesis, $W; \Gamma, y : \bar{\tau} \vdash^{\Sigma} \{x \mapsto X_1\}\bar{M} \hookrightarrow \{x \mapsto X_2\}\bar{N} : \bar{s}, \bar{\sigma}'$ with $\bar{\sigma} \preccurlyeq \bar{\sigma}'$. By $\textsc{Abs}_\mathrm{I}$, $\Gamma \vdash^{\Sigma} (\lambda y.\{x \mapsto X_1\}\bar{M}) \hookrightarrow (\lambda y.\{x \mapsto X_2\}\bar{N}) : s, \bar{\tau} \xrightarrow{\bar{s}} \bar{\sigma}'$, and we take $\tau' = \bar{\tau} \xrightarrow{\bar{s}} \bar{\sigma}'$.

Rec$_\mathbf{I}$. Here $M = (\varrho y.\bar{X}_1)$, $M = (\varrho y.\bar{X}_2)$, by Remark Appendix B.13 we have $s = \mho$, and $W; \Gamma, x : \sigma, y : \tau \vdash^{\Sigma} \bar{X}_1 \hookrightarrow \bar{X}_2 : s, \tau$. We can assume that $y \notin \mathrm{dom}(W; \Gamma, x : \sigma)$ (otherwise rename y). Then $\{x \mapsto X_1\}(\varrho y.\bar{X}_1) = (\varrho y.\{x \mapsto X_1\}\bar{X}_1)$ and $\{x \mapsto X_2\}(\varrho y.\bar{X}_2) = (\varrho y.\{x \mapsto X_2\}\bar{X}_2)$. By assumption and Lemma Appendix B.14 we have $W; \Gamma, y : \tau \vdash^{\Sigma} X_1 \hookrightarrow X_2 : s, \sigma'$. By induction hypothesis, $W; \Gamma, y : \tau \vdash^{\Sigma} \{x \mapsto X_1\}\bar{X}_1 \hookrightarrow \{x \mapsto X_2\}\bar{X}_2 : \mho, \tau'$, with $tau \preccurlyeq \tau'$. Then, by $\textsc{Rec}_\mathrm{I}$, $\Gamma \vdash^{\Sigma} (\varrho y.\{x \mapsto X_1\}\bar{X}_1) \hookrightarrow (\varrho y.\{x \mapsto X_2\}\bar{X}_2) : \mho, \tau'$.

Ref$_\mathbf{I}$. Here $M = (\mathrm{ref}_\theta\ \bar{M})$, $N = (\mathrm{ref}_\theta\ \bar{N})$ and we have $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \theta'$ where $\theta \preccurlyeq \theta'$ and $\tau = \theta\ \mathrm{ref}$. By induction hypothesis, $\Gamma \vdash^{\Sigma} \{x \mapsto X_1\}\bar{M} \hookrightarrow \{x \mapsto X_2\}\bar{N} : \bar{s}, \theta''$ with $\theta' \preccurlyeq \theta''$. Then, $\theta \preccurlyeq \theta''$, so we conclude by $\textsc{Ref}_\mathrm{I}$ that $\Gamma \vdash^{\Sigma} (\mathrm{ref}_\theta\ \{x \mapsto X_1\}\bar{M}) \hookrightarrow (\mathrm{ref}_\theta\ \{x \mapsto X_2\}\bar{N}) : s, \tau$.

Cond$_\mathbf{I}$. Here $M = (\text{if } \bar{M} \text{ then } \bar{M}_t \text{ else } \bar{M}_f)$, $N = (\text{if } \bar{N} \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$ and we have $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M} \hookrightarrow \bar{N} : \bar{s}, \mathsf{bool}$, $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M}_t \hookrightarrow \bar{N}_t : \bar{s_t}, \bar{\tau_t}$ and $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M}_f \hookrightarrow \bar{N}_f : \bar{s_f}, \bar{\tau_f}$ with $s = \bar{s} \curlywedge \bar{s_t} \curlywedge \bar{s_f}$, $\bar{\tau_t} \approx \bar{\tau_f}$ and $\tau = \bar{\tau_t} \curlywedge \bar{\tau_f}$. By induction hypothesis, $W; \Gamma, x : \sigma \vdash^\Sigma \{x \mapsto X_1\}\bar{M} \hookrightarrow \{x \mapsto X_2\}\bar{N} : \bar{s}, \mathsf{bool}$, $W; \Gamma, x : \sigma \vdash^\Sigma \{x \mapsto X_1\}\bar{M}_t \hookrightarrow \{x \mapsto X_2\}\bar{N}_t : \bar{s_t}, \bar{\tau_t}'$ and $W; \Gamma, x : \sigma \vdash^\Sigma \{x \mapsto X_1\}\bar{M}_f \hookrightarrow \{x \mapsto X_2\}\bar{N}_f : \bar{s_f}, \bar{\tau_f}$' with $\bar{\tau_t} \preccurlyeq \bar{\tau_t}'$ and $\bar{\tau_f} \preccurlyeq \bar{\tau_f}'$. It is still the case that $\bar{\tau_t}' \approx \bar{\tau_f}'$, so by rule Cond$_\mathrm{I}$ we have that $W; \Gamma, x : \sigma \vdash^\Sigma (\text{if } \{x \mapsto X_1\}\bar{M} \text{ then } \{x \mapsto X_1\}\bar{M}_t \text{ else } \{x \mapsto X_1\}\bar{M}_f) \hookrightarrow (\text{if } \{x \mapsto X_2\}\bar{N} \text{ then } \{x \mapsto X_2\}\bar{N}_t \text{ else } \{x \mapsto X_2\}\bar{N}_f) : s, \bar{\tau_t}' \curlywedge \bar{\tau_f}'$. We then take $\tau' = \bar{\tau_t}' \curlywedge \bar{\tau_f}'$.

App$_\mathbf{I}$. Here $M = (\bar{M}_1\ \bar{M}_2)$ and we have that $\Gamma \vdash^\Sigma \bar{M}_1 \hookrightarrow \bar{N}_1 : \bar{s_1}, \bar{\theta} \xrightarrow{\bar{s_3}} \bar{\sigma}$ and $\Gamma \vdash^\Sigma \bar{M}_2 \hookrightarrow \bar{N}_2 : \bar{s_2}, \bar{\theta}''$ where $s = \bar{s_1} \curlywedge \bar{s_2} \curlywedge \bar{s_3}$, $\bar{\theta} \preccurlyeq \bar{\theta}''$, and $\tau = \bar{\sigma}$. By induction hypothesis, $W; \Gamma, x : \sigma \vdash^\Sigma \{x \mapsto X_1\}\bar{M}_1 \hookrightarrow \{x \mapsto X_2\}\bar{N}_1 : \bar{s_t}, \bar{\theta} \xrightarrow{\bar{s_3}'} \bar{\sigma}'$ and $W; \Gamma, x : \sigma \vdash^\Sigma \{x \mapsto X_2\}\bar{M}_f \hookrightarrow \{x \mapsto X_2\}\bar{N}_2 : \bar{s_f}, \bar{\theta}'''$ with $\bar{s_3} \preccurlyeq \bar{s_3}'$, $\bar{\sigma} \preccurlyeq \bar{\sigma}'$ and $\bar{\theta}'' \preccurlyeq \bar{\theta}'''$. It is still the case that $\bar{\theta} \preccurlyeq \bar{\theta}'''$. Therefore, by rule App$_\mathrm{I}$ we have that $W; \Gamma, x : \sigma \vdash^\Sigma (\{x \mapsto X_1\}\bar{M}_1\ \{x \mapsto X_1\}\bar{M}_2) \hookrightarrow (\text{if } \{x \mapsto X_2\}\bar{M}_2 \text{ then } \{x \mapsto X_2\}\bar{N}_t \text{ else } \{x \mapsto X_2\}\bar{N}_2) : s, \bar{\sigma}'$, and we take $\tau' = \bar{\sigma}'$.

Mig$_\mathbf{I}$. Here $M = (\text{thread}_l\ \bar{M} \text{ at } d)$, $N = (\text{thread}_l^{\bar{s}}\ \bar{N} \text{ at } d)$, $s = \mho$ and we have that $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M} \hookrightarrow \bar{N} : \bar{s}, \bar{\tau}$ and $\tau = \mathsf{unit}$. By induction hypothesis, then $\Gamma \vdash^\Sigma \{x \mapsto X_1\}\bar{M} \hookrightarrow \{x \mapsto X_2\}\bar{N} : \bar{s}, \bar{\tau}$. Therefore, by rule Mig$_\mathrm{I}$, $\Gamma \vdash^\Sigma (\text{thread}_l\ \{x \mapsto X_1\}\bar{M} \text{ at } d) \hookrightarrow (\text{thread}_l^{\bar{s}}\ \{x \mapsto X_2\}\bar{N} \text{ at } d) : \mho, \tau$.

Flow$_\mathbf{I}$. Here $M = (\text{flow } \bar{F} \text{ in } \bar{M})$, $N = (\text{flow } \bar{F} \text{ in } \bar{N})$ and $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M} \hookrightarrow \bar{N} : \bar{s}, \tau$ with $s = \bar{s} \curlywedge \bar{F}$. By induction hypothesis, $\Gamma \vdash^\Sigma \{x \mapsto X_1\}\bar{M} \hookrightarrow \{x \mapsto X_2\}\bar{N} : \bar{s}, \tau'$ with $\tau \preccurlyeq \tau'$. By Flow$_\mathrm{I}$, $\Gamma \vdash^\Sigma (\text{flow } \bar{F} \text{ in } \{x \mapsto X_1\}\bar{M}) \hookrightarrow (\text{flow } \bar{F} \text{ in } \{x \mapsto X_2\}\bar{N}) : s, \tau'$.

Allow$_\mathbf{I}$. Here we have that $M = (\text{allowed } \bar{F} \text{ then } \bar{M}_t \text{ else } \bar{M}_f)$, and that $N = (\text{allowed } \bar{F} \text{ then } \bar{N}_t \text{ else } \bar{N}_f)$ and $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M} \hookrightarrow \bar{M} : \bar{s}, \mathsf{bool}$ and also $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M}_t \hookrightarrow \bar{N}_t : \bar{s_t}, \bar{\tau_t}$ and $W; \Gamma, x : \sigma \vdash^\Sigma \bar{M}_f \hookrightarrow \bar{N}_f : \bar{s_f}, \bar{\tau_f}$, with $s = \bar{s_t} \curlyvee \bar{F} \curlywedge \bar{s_f}$, $\bar{\tau_t} \approx \bar{\tau_f}$ and $\tau = \bar{\tau_t} \curlywedge \bar{\tau_f}$. By induction hypothesis, $\Gamma \vdash^\Sigma \{x \mapsto X_1\}\bar{M}_t \hookrightarrow \{x \mapsto X_2\}\bar{N}_t : \bar{s_t}, \bar{\tau_t'}$ and $\Gamma \vdash^\Sigma \{x \mapsto X_1\}\bar{M}_f \hookrightarrow \{x \mapsto X_2\}\bar{N}_f : \bar{s_f}, \bar{\tau_f'}$ with $\bar{\tau_t} \preccurlyeq \bar{\tau_t'}$ and $\bar{\tau_f} \preccurlyeq \bar{\tau_f'}$. Still, $\bar{\tau_t'} \approx \bar{\tau_f'}$. By Allow$_\mathrm{I}$, we have that $\Gamma \vdash^\Sigma (\text{allowed } \bar{F} \text{ then } \{x \mapsto X_1\}\bar{M}_t \text{ else } \{x \mapsto X_1\}\bar{M}_f) \hookrightarrow (\text{allowed } \bar{F} \text{ then } \{x \mapsto X_1\}\bar{N}_t \text{ else } \{x \mapsto X_1\}\bar{N}_f) : s, \bar{s_t}' \curlyvee \bar{F} \curlywedge \bar{s_f}'$. We then take $\tau' = \bar{\tau_t} \curlywedge \bar{\tau_f}$.

The proofs for the cases Loc$_\mathrm{I}$, Bt$_\mathrm{I}$ and Bf$_\mathrm{I}$ are analogous to the one for Nil$_\mathrm{I}$, while the proofs for Seq$_\mathrm{I}$, Der$_\mathrm{I}$ and Ass$_\mathrm{I}$ are analogous to (or simpler than) the one for App$_\mathrm{I}$.

□

Lemma Appendix B.16 (Replacement)**.**

1. *If* $\Gamma \vdash^{\Sigma} \mathrm{E}_1[M] \hookrightarrow N_{E_2} : s, \tau$ *is a valid judgment, then the proof gives* M *a typing* $\Gamma \vdash^{\Sigma} M \hookrightarrow N : \bar{s}, \bar{\tau}$ *for some* M, $\bar{s}$ *and* $\bar{\tau}$ *such that* $s \preccurlyeq \bar{s}$ *and for which there exists* E_2 *such that* $M_{E_2} = \mathrm{E}_1[M]$. *In this case, if* $\Gamma \vdash^{\Sigma} M' \hookrightarrow N' : \bar{s}', \bar{\tau}'$ *with* $\bar{s} \curlywedge A \preccurlyeq \bar{s}'$ *for some* A *and* $\bar{\tau} \preccurlyeq \bar{\tau}'$, *then* $\Gamma \vdash^{\Sigma} \mathrm{E}_1[M'] \hookrightarrow \mathrm{E}_2[N'] : s', \tau'$ *for some* s', τ' *such that* $s \curlywedge A \preccurlyeq s'$ *and* $\tau \preccurlyeq \tau'$.
2. *If* $\Gamma \vdash^{\Sigma} M_{E_1} \hookrightarrow \mathrm{E}_2[N] : s, \tau$ *is a valid judgment, then the proof gives* N *a typing* $\Gamma \vdash^{\Sigma} M \hookrightarrow N : \bar{s}, \bar{\tau}$ *for some* M, $\bar{s}$ *and* $\bar{\tau}$ *such that* $s \preccurlyeq \bar{s}$ *and for which there exists* E_1 *such that* $M_{E_1} = \mathrm{E}_1[M]$. *In this case, if* $\Gamma \vdash^{\Sigma} M' \hookrightarrow N' : \bar{s}', \bar{\tau}'$ *with* $\bar{s} \curlywedge A \preccurlyeq \bar{s}'$ *for some* A *and* $\bar{\tau} \preccurlyeq \bar{\tau}'$, *then* $\Gamma \vdash^{\Sigma} \mathrm{E}_1[M'] \hookrightarrow \mathrm{E}_2[N'] : s', \tau'$ *for some* s', τ' *such that* $s \curlywedge A \preccurlyeq s'$ *and* $\tau \preccurlyeq \tau'$.

Proof. 1. By induction on the structure of E_1. The proof is "symmetric" to the second case.

2. By induction on the structure of E_2.

   $\mathrm{E}_2[N] = N$. This case is direct.

   $\mathrm{E}_2[N] =$ (if $\hat{\mathrm{E}}_2[N]$ then $\hat{N}_t$ else $\hat{N}_f$). By rule $\textsc{Cond}_{\mathrm{I}}$, we have that $M_{E_1} = (\text{if } M_{\hat{E}_1} \text{ then } \hat{M}_t \text{ else } \hat{M}_f)$, and $\Gamma \vdash^{\Sigma} M_{\hat{E}_1} \hookrightarrow \hat{\mathrm{E}}_2[N] : \hat{s}, \mathsf{bool}$, and also $\Gamma \vdash^{\Sigma} \hat{M}_t \hookrightarrow \hat{N}_t : \hat{s_t}, \hat{\tau_t}$ and $\Gamma \vdash^{\Sigma} \hat{M}_f \hookrightarrow \hat{N}_f : \hat{s_f}, \hat{\tau_f}$ with $s = \hat{s} \curlywedge \hat{s_t} \curlywedge \hat{s_f}$, $\hat{\tau_t} \approx \hat{\tau_f}$ and $\tau = \hat{\tau_t} \curlywedge \hat{\tau_f}$. By induction hypothesis, the proof gives N a typing $\Gamma \vdash^{\Sigma} M \hookrightarrow N : \bar{s}, \bar{\tau}$, for some M, $\bar{s}$, $\bar{\tau}$ such that $\hat{s} \preccurlyeq \bar{s}$, and for which there exists $\hat{\mathrm{E}}_1$ such that $M_{\hat{E}_1} = \hat{\mathrm{E}}_1[M]$.

   Also by induction hypothesis, $\Gamma \vdash^{\Sigma} \hat{\mathrm{E}}_1[M'] \hookrightarrow \hat{\mathrm{E}}_2[N'] : \hat{s}', \mathsf{bool}$, for some $\hat{s}'$ such that $\hat{s} \curlywedge A \preccurlyeq \hat{s}'$. Again by rule $\textsc{Cond}_{\mathrm{I}}$, $\Gamma \vdash^{\Sigma} (\text{if } \hat{\mathrm{E}}_1[M'] \text{ then } \hat{M}_t \text{ else } \hat{M}_f) \hookrightarrow (\text{if } \hat{\mathrm{E}}_2[N'] \text{ then } \hat{N}_t \text{ else } \hat{N}_f) : s', \tau'$ with $s' = \hat{s}' \curlywedge \hat{s_t} \curlywedge \hat{s_f}$ and $\tau' = \hat{\tau_t} \curlywedge \hat{\tau_f}$. Notice that $s \curlywedge A = \hat{s} \curlywedge \hat{s_t} \curlywedge \hat{s_f} \curlywedge A \preccurlyeq \hat{s}' \curlywedge \hat{s_t} \curlywedge \hat{s_f} = s'$.

   $\mathrm{E}_2[M] =$ (flow $\hat{F}$ in $\hat{\mathrm{E}}_2[M]$). By $\textsc{Flow}_{\mathrm{I}}$, $M_{E_1} = (\text{flow } \hat{F} \text{ in } M_{\hat{E}_1})$, and $\Gamma \vdash^{\Sigma} M_{\hat{E}_1} \hookrightarrow \hat{\mathrm{E}}_2[N] : \hat{s}, \tau$ and $s = \hat{s} \curlywedge \hat{F}$. By induction hypothesis, the proof gives N a typing $\Gamma \vdash^{\Sigma} M \hookrightarrow N : \bar{s}, \bar{\tau}$, for some M, $\bar{s}$, $\bar{\tau}$ such that $\hat{s} \preccurlyeq \bar{s}$, and for which there exists $\hat{\mathrm{E}}_1$ such that $M_{\hat{E}_1} = \hat{\mathrm{E}}_1[M]$.

   Also by induction hypothesis, $\Gamma \vdash^{\Sigma} \hat{\mathrm{E}}_1[M'] \hookrightarrow \hat{\mathrm{E}}_2[N'] : \hat{s}', \tau$, for some $\hat{s}'$ such that $\hat{s} \curlywedge A \preccurlyeq \hat{s}'$. Then, again by $\textsc{Flow}_{\mathrm{I}}$, we have $\Gamma \vdash^{\Sigma} (\text{flow } \hat{F} \text{ in } \hat{\mathrm{E}}_1[M']) \hookrightarrow (\text{flow } \bar{F} \text{ in } \hat{\mathrm{E}}_2[N']) : s', \tau$ with $s' = \hat{s}' \curlywedge \hat{F}$. Notice that $s \curlywedge A = \hat{s} \curlywedge \hat{F} \curlywedge A \preccurlyeq \hat{s}' \curlywedge \hat{F} = s'$.

   The proofs for the cases $\mathrm{E}_2[M] = \lceil(\hat{\mathrm{E}}_2[M] := N)\rceil$, $\mathrm{E}_2[M] = \lceil(V := \hat{\mathrm{E}}_2[M])\rceil$, $\mathrm{E}_2[M] = \lceil(!\ \hat{\mathrm{E}}_2[M])\rceil$, $\mathrm{E}_2[M] = \lceil(\hat{\mathrm{E}}_2[M]\ N)\rceil$, $\mathrm{E}_2[M] = \lceil(V\ \hat{\mathrm{E}}_2[M])\rceil$, $\mathrm{E}_2[M] = \lceil(\hat{\mathrm{E}}_2[M]; N)\rceil$ and $\mathrm{E}_2[M] = \lceil(\mathrm{ref}_{l,\theta}\ \hat{\mathrm{E}}_2[M])\rceil$, are all analogous to the proof for the case $\mathrm{E}_2[M] = (\text{if } \hat{\mathrm{E}}_2[M] \text{ then } N_t \text{ else } N_f)$.

□

The following proposition ensures that the annotation processing is preserved by the annotated semantics. This is formulated by stating that after reduction,

programs are still well annotated. More precisely, the following result states that if a program is the result of an annotation process, a certain declassification effect and type, then after one computation step it is still the result of annotating a program, and is given a not-more permissive declassification effect and type.

Proposition Appendix B.17 (Subject Reduction, or Preservation of Annotations)**.** *Given an allowed-policy mapping W, a reference labeling Σ and a typing environment Γ, consider a thread M^m for which there exist N, s and τ such that $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$ and suppose that $W \vdash^{\Sigma,\Upsilon} \langle\{N^m\}, T, S\rangle \xrightarrow[F]{d} \langle\{N'^m\} \cup P, T', S'\rangle$, for a memory S that is (Σ,Γ)-compatible. Then there exist M', s', τ' such that $s \curlywedge W(T(m)) \preccurlyeq s'$, and $\tau \preccurlyeq \tau'$, and $\Gamma \vdash^{\Sigma} M' \hookrightarrow N' : s', \tau'$, and S' is also (Σ,Γ)-compatible. Furthermore, if $P = \{N''^n\}$ for some expression N'' and thread name n, then there exist M'', s'' such that $W(T'(n)) \preccurlyeq s''$ and $\Gamma \vdash^{\Sigma} M'' \hookrightarrow N'' : s'', \mathsf{unit}$.*

Proof. Suppose that $N = \bar{\mathrm{E}}[\bar{N}]$ and $W \vdash^{\Sigma,\Upsilon} \langle\{\bar{N}^m\}, T, S\rangle \xrightarrow[\bar{F}]{d} \langle\{\bar{N}'^m\} \cup P', \bar{T}', \bar{S}'\rangle$. We start by observing that this implies $F = \bar{F} \curlywedge \lceil\bar{\mathrm{E}}\rceil$, $M' = \bar{\mathrm{E}}[\bar{M}']$, $P = P'$, $\bar{T}' = T'$ and $\bar{S}' = S'$. We can assume, without loss of generality, that $\bar{N}$ is the smallest in the sense that there is no $\hat{\mathrm{E}}, \hat{N}$ such that $\hat{\mathrm{E}} \neq []$ and $\hat{\mathrm{E}}[\hat{N}] = \bar{N}$ for which we can write $W \vdash^{\Sigma,\Upsilon} \langle\{\hat{N}^m\}, T, S\rangle \xrightarrow[\hat{F}]{d} \langle\{\hat{N}'^m\} \cup P, T', S'\rangle$.

By Lemma Appendix B.16, we have $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \bar{\tau}$, for some $\bar{M}$, $\bar{s}$' $\bar{\tau}$ such that $s \preccurlyeq \bar{s}$, in the proof of $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$. We proceed by case analysis on the transition $W \vdash^{\Sigma,\Upsilon} \langle\{\bar{N}^m\}, T, S\rangle \xrightarrow[\bar{F}]{d} \langle\{\bar{N}'^m\} \cup P, T', S'\rangle$, and prove that:

- There exist $\bar{M}'$, $\bar{s}'$ and $\bar{\tau}'$ such that $\Gamma \vdash^{\Sigma} \bar{M}' \hookrightarrow \bar{N}' : \bar{s}', \bar{\tau}'$, and $\bar{s} \curlywedge W(T(m)) \preccurlyeq \bar{s}'$ and $\bar{\tau} \preccurlyeq \bar{\tau}'$. Furthermore, for every reference $a \in \mathrm{dom}(S')$ implies $\Gamma \vdash^{\Sigma}_{\mho,} V \hookrightarrow S'(a) : \Sigma_2(a)$ for some value V.
- If $P = \{N''^n\}$ for some expression N'' and thread name n, then there exist M'' and $\bar{s}''$ such that $\Gamma \vdash^{\Sigma} M'' \hookrightarrow N'' : \bar{s}'', \mathsf{unit}$, and $W(T'(n)) \preccurlyeq \bar{s}'$. (Note that in this case $S = S'$.)

By case analysis on the structure of $\bar{N}$:

$\bar{N} = ((\lambda x.\hat{N})\ V_2)$. Here we have $\bar{N}' = \{x \mapsto V_2\}\hat{N}$, $S = S'$ and $P = \emptyset$. By rule $\textsc{App}_{\mathrm{I}}$, there exist $\hat{M}$, V_1, $\hat{s_1}$, $\hat{s_2}$, $\hat{s_3}$, $\hat{\tau}$, $\hat{\sigma}$ and $\hat{\tau}''$ such that $\Gamma \vdash^{\Sigma} (\lambda x.\hat{M}) \hookrightarrow (\lambda x.\hat{N}) : \hat{s_1}, \hat{\tau} \xrightarrow{\hat{s_3}} \hat{\sigma}$ and $\Gamma \vdash^{\Sigma} V_1 \hookrightarrow V_2 : \hat{s_2}, \hat{\tau}''$ with $\bar{s} = \hat{s_1} \curlywedge \hat{s_2} \curlywedge \hat{s_3}$, $\hat{\tau} \preccurlyeq \hat{\tau}''$ and $\bar{\tau} = \hat{\sigma}$. By $\textsc{Abs}_{\mathrm{I}}$, then $\Gamma, x : \hat{\tau} \vdash^{\Sigma} \hat{M} \hookrightarrow \hat{N} : \hat{s_3}, \hat{\sigma}$. Therefore, by Lemma Appendix B.15, we get $\Gamma \vdash^{\Sigma} \{x \mapsto V_1\}\hat{M} \hookrightarrow \{x \mapsto V_2\}\hat{N} : \hat{s_3}', \hat{\sigma}'$ with $\hat{s_3} \preccurlyeq \hat{s_3}'$ and $\hat{\sigma} \preccurlyeq \bar{\tau}'$. We take $\bar{s}' = \hat{s_3}$ and $\bar{\tau}' = \hat{\sigma}'$.

$\bar{N} =$ (flow $\hat{F}$ in V_2). Here we have $\bar{N}' = V_2$, $S = S'$ and $P = \emptyset$. By rule $\textsc{Flow}_{\mathrm{I}}$ and by Remark Appendix B.12, there exist V_1, $\hat{s}$, such that $\Gamma \vdash^{\Sigma} V_1 \hookrightarrow V_2 : \mho, \bar{\tau}$. We take $\bar{s}' = \mho$.

$\bar{N}$ = (allowed $\hat{F}$ then N_t else N_f) and $W(T(m)) \preccurlyeq \hat{F}$. Here we have $\bar{N}' = N_t$, $S = S'$ and $P = \emptyset$. By $\textsc{Allow}_{\text{I}}$, there exist M_t, N_t, $\hat{s_t}$, $\hat{s_f}$, $\hat{\tau_t}$ and $\hat{\tau_f}$ such that $\Gamma \vdash^{\Sigma} M_t \hookrightarrow N_t : \hat{s_t}, \hat{\tau_t}$, and $\Gamma \vdash^{\Sigma} M_f \hookrightarrow N_f : \hat{s_f}, \hat{\tau_f}$, where $\bar{s} = \hat{s_t} \smile \hat{F} \curlywedge \hat{s_f}$, $\tau_t \approx \tau_f$ and $\bar{\tau} = \hat{\tau_t} \curlywedge \hat{\tau_f}$. We take $\bar{s}' = \hat{s_t}$ and $\bar{\tau}' = \hat{\tau_t}$. Notice that $\bar{s} \curlywedge W(T(m)) = \hat{s_t} \smile \hat{F} \curlywedge \hat{s_f} \curlywedge W(T(m)) \preccurlyeq \hat{s_t}$ and $\bar{\tau} \preccurlyeq \bar{\tau}'$.

$\bar{N}$ = ($\text{thread}^{\dot{s}}_{k}$ $\hat{N}$ at d). Here we have $\bar{N}' = ()$, $S = S'$, $P = \{\hat{N}^n\}$ for some thread name n, $T'(n) = d$ and $W(d) \preccurlyeq \dot{s}$. By $\textsc{Mig}_{\text{I}}$, we have that there exists $\hat{M}$ such that $\Gamma \vdash^{\Sigma} \hat{M} \hookrightarrow \hat{N} : \dot{s}, \mathsf{unit}$ and $\bar{\tau} = \mathsf{unit}$, and by $\textsc{Nil}_{\text{I}}$ we have that $\Gamma \vdash^{\Sigma} () \hookrightarrow () : \mho, \mathsf{unit}$.

By Lemma Appendix B.16, we can finally conclude that $\Gamma \vdash^{\Sigma} M'_E \hookrightarrow \bar{\text{E}}[\bar{N}'] : s', \tau'$ for some M'_E, s', τ' such that $s \curlywedge W(T(m)) \preccurlyeq s'$ and $\tau \preccurlyeq \tau'$.

□

Appendix B.4.2. Preservation of the semantics

Proposition Appendix B.18. *Consider a given a typing environment Γ and reference labeling Σ. If there exist s, τ such that $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$, then for all thread names $m \in \textbf{\textit{Nam}}$ we have that $\{M^m\} \sim_{\Gamma} \{N^m\}$.*

Proof. We prove that the set

$$B = \{\langle \{M^m\}, \{N^m\} \rangle \mid m \in \textbf{\textit{Nam}} \text{ and } \exists s, \tau \,.\, \Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau\}$$

is a (W, Σ, Γ)-simulation according to Definition 4.14.

Suppose that $N = \bar{\text{E}_2}[\bar{N}]$ and $W \vdash^{\Sigma, \Upsilon} \langle \{\bar{N}^m\}, T, \hat{S} \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{N}'^m\} \cup \bar{P}'_2, \bar{T}', \bar{\hat{S}}' \rangle$. We start by observing that this implies $F = \bar{F} \curlywedge \lceil \bar{\text{E}_2} \rceil$, $M' = \bar{\text{E}_2}[\bar{M}']$, $P'_2 = \bar{P}'_2$, $\bar{T}' = T'$ and $\bar{\hat{S}}' = \hat{S}'$. We can assume, without loss of generality, that $\bar{N}$ is the smallest in the sense that there is no $\hat{\text{E}_2}, \hat{N}$ such that $\hat{\text{E}_2} \neq []$ and $\hat{\text{E}_2}[\hat{N}] = \bar{N}$ for which we can write $W \vdash^{\Sigma, \Upsilon} \langle \{\hat{N}^m\}, T, \hat{S} \rangle \xrightarrow[\hat{F}]{d} \langle \{\hat{N}'^m\} \cup P'_2, T', \hat{S}' \rangle$.

By Lemma Appendix B.16, we have $\Gamma \vdash^{\Sigma} \bar{M} \hookrightarrow \bar{N} : \bar{s}, \bar{\tau}$, for some $\bar{M}$, $\bar{s}$' $\bar{\tau}$ such that $s \preccurlyeq \bar{s}$, in the proof of $\Gamma \vdash^{\Sigma} M \hookrightarrow N : s, \tau$. Furthermore, there exists $\bar{\text{E}}_1$ such that $M = \bar{\text{E}}_1[\bar{M}]$. We proceed by case analysis on the transition $W \vdash^{\Sigma, \Upsilon} \langle \{\bar{N}^m\}, T, \hat{S} \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{N}'^m\} \cup P'_2, T', \hat{S}' \rangle$, and prove that:

- there exist $\bar{M}'$, S' such that $W \vdash^{\Sigma, \Upsilon} \langle \{\bar{M}^m\}, T, S \rangle \xrightarrow[\bar{F}]{d} \langle \{\bar{M}'^m\} \cup P'_1, T', S' \rangle$ and $\text{annot}(S') = \hat{S}'$, and for which there exist $\bar{s}'$ and $\bar{\tau}'$ such that $\Gamma \vdash^{\Sigma} \bar{M}' \hookrightarrow \bar{N}' : \bar{s}', \bar{\tau}'$.
- If $P'_2 = \{N''^n\}$ for some expression N'' and thread name n, then there exist M'' and s'' such that $\Gamma \vdash^{\Sigma} M'' \hookrightarrow N'' : s'', \mathsf{unit}$.

By case analysis on the structure of $\bar{N}$:

$\bar{N}$ = (flow $\hat{F}$ in V_2). Here we have $\bar{N}' = V_2$, $S = S'$ and $P_2' = \emptyset$. By rule $\textsc{Flow}_\textsc{I}$ and Remark Appendix B.12, there exist V_1, $\hat{s}$, such that $\bar{M} = (\text{flow } \hat{F} \text{ in } V_1)$ and $\Gamma \vdash^\Sigma V_1 \hookrightarrow V_2 : \hat{s}, \bar{\tau}$. We then have $W \vdash \langle \{\bar{M}^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{{V_1}^m\}, T, S\rangle$, so we take $\bar{M}' = V_1$.

$\bar{N}$ = (allowed $\hat{F}$ then N_t else N_f) and $W(T(m)) \preccurlyeq \hat{F}$. Here we have that $\bar{N}' = N_t$, $S = S'$ and $P_2' = \emptyset$. By $\textsc{Allow}_\textsc{I}$, there exist M_t, M_f, $\hat{s_t}$, $\hat{s_f}$, $\hat{\tau_t}$ and $\hat{\tau_f}$ such that $\bar{M} = (\text{allowed } \hat{F} \text{ then } M_t \text{ else } M_f)$ and $\Gamma \vdash^\Sigma M_t \hookrightarrow N_t : \hat{s_t}, \hat{\tau_t}$, and $\Gamma \vdash^\Sigma M_f \hookrightarrow N_f : \hat{s_f}, \hat{\tau_f}$, where $\bar{s} = \hat{s_t} \smile \hat{F} \curlywedge \hat{s_f}$, $\tau_t \approx \tau_f$ and $\bar{\tau} = \hat{\tau_t} \curlywedge \hat{\tau_f}$. We take $\bar{s}' = \hat{s_t}$ and $\bar{\tau}' = \hat{\tau_t}$. We then have $W \vdash \langle \{\bar{M}^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{{M_t}^m\}, T, S\rangle$, so we take $\bar{M}' = M_t$.

$\bar{N}$ = ($\text{thread}_k^{\dot{s}}$ $\hat{N}$ at d). Here we have $\bar{N}' = ()$, $S = S'$, $P = \{\hat{N}^n\}$ for some thread name n, $T'(n) = d$ and $W(d) \preccurlyeq \dot{s}$. By $\textsc{Mig}_\textsc{I}$, we have that there exists $\hat{M}$ such that $\bar{M} = (\text{thread}_k\ \hat{N} \text{ at } d)$ and $\Gamma \vdash^\Sigma \hat{M} \hookrightarrow \hat{N} : \dot{s}, \mathsf{unit}$ and $\bar{\tau} = \mathsf{unit}$, and by $\textsc{Nil}_\textsc{I}$ we have that $\Gamma \vdash^\Sigma () \hookrightarrow () : \mho, \mathsf{unit}$. By Proposition 4.13 we have that. Therefore, $W \vdash \langle \{\bar{M}^m\}, T, S\rangle \xrightarrow[F]{d} \langle \{()^m\}, T, S\rangle$, so we take $\bar{M}' = ()$.

By Lemma Appendix B.16, we can finally conclude that $\Gamma \vdash^\Sigma \bar{\mathrm{E}}_1[\bar{M}'] \hookrightarrow \bar{\mathrm{E}}_2[\bar{N}'] : s', \tau'$ for some s', τ'.

□