Flow-policy awareness for distributed mobile code

Ana Almeida Matos
Instituto Superior Técnico de Lisboa and Instituto de Telecomunicações
ana.matos@ist.utl.pt

Abstract

Several programming constructs have recently been proposed with the purpose of enabling the programmer to encode declassifying information flows within a program that complies with information flow security policies. These constructs may or may not incorporate some means for controlling when, where, what, or by whom the declassification can be set up. In the context of global computing, other forms of controlling declassification that transcend the power of a single declassification construct may turn out to be desirable. In this paper we point out potential unwanted behaviors that can arise in a context where programs that contain declassifying instructions can migrate to computation domains with different security policies. We propose programming language design techniques for tackling such unwanted behaviors and prove soundness of those techniques at the global computation level.

1. Introduction

The new possibilities opened by global computing have brought information security issues to a new level of concern. Indeed, such possibilities can just as well be exploited by parties with hazardous intentions. Many attacks arise at the application level, and can be tackled by means of programming language design and analysis techniques, such as static analysis and proof carrying code. For instance, confidentiality can be violated by execution of programs that reveal secret information. This kind of program behavior can be controlled using *information flow* analyses [22], by detecting dependencies in programs that could encode flows of information from private to publicly available resources.

In the field of security, control must often be balanced with flexibility for practical reasons, since models that are prohibitively restrictive do not suit the real world-needs. In information flow research, it has been a challenging problem to find an alternative to the classical non-interference property [11] that is flexible enough to allow for *declassification* to take place in a controlled manner [26]. So far, most solutions 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 [23], since now the question is: *in which context?*

In this paper we show that the issue of enabling and controlling flexible information flow policies in computations that can spread out over sites that are governed by different flow policies 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. Given the appropriate tools to predict alternatives to the pieces of code that contain potentially forbidden declassification operations, it becomes realistic to write programs that can safely run under any flow policy.

Some security minded distributed network models have been proposed with the purpose of controlling the migration of code in between computation sites, such as by means of programmable domains [4] and type systems [17]. These ideas can be applied to the proof-carrying code model [20], since it consists of a particular instance of boundary transposition control that performs type checks to incoming code [12]. We propose 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 of 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.

Setting We follow the non-disclosure point of view of information flow analysis presented in [1, 2]. 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 to the valid flow policy. In order to enable dynamic changes to the valid flow policy, the programming language may be enriched with a *flow declaration* construct $(\mathrm{flow}\ F\ \mathrm{in}\ M)$ that simply declares the flow policy (F) that is valid in its scope (M) within the program, while the semantics of the language may convey information regarding the flow policy that rules at each step. 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:

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

In this example, the program declares that flows in M comply to a flow policy that is extended (made more permissive) by $F_{\textit{permissive}}$ (the N branch is of course not given this flexibility). In other words, M may contain declassifications that comply to $F_{\textit{permissive}}$.

Once the language is enriched with flow declarations (or any other means for expressing declassification), some mechanism for controlling the usage of that construct is desirable. This is particularly relevant in distributed settings with mobile code. For instance, a computation domain d might want to impose a limit to the flexibility of the flow declarations that are used within its programs, since so far nothing prevents incoming code from containing:

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

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 to 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 that the program must comply to. 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 $(\mathrm{allowed}\ F\ \mathrm{then}\ M\ \mathrm{else}\ N)$, 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:

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

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, as we will see right ahead, misbehaving programs can be rejected by the domains where they would like to execute, so it is in the interest of the programmer to increase the chances that its program will be allowed to execute, by adequately "protecting" portions of code containing declassifications by appropriate allowed-conditions.

In the spirit of the proof carrying code model, 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 that tests the declared flow policy. We call this flow policy the *declassification effect* of the program, and provide a type system for obtaining it. Then, while the program

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

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

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

(assuming that M and N do not contain any flow declarations) would have an empty declassification effect – meaning that it could be safely allowed to run in any domain.

In order to formalize these ideas, it is useful to consider a concrete distributed language with code mobility, where we use the declassification effect to control migration according to the following rule: programs can only migrate to a site if their declassification behaviors comply to that site's flow policy. We can then analyze the conditions under which the programs of this distributed language comply to a network level version of the information flow and confinement properties.

Outline of the paper We start by introducing the language and its operational semantics (Section 2). Two main section follow, each presenting the security analysis for our information flow property of Non-disclosure (Section 3) and for the new Confinement property (Section 4). Each of these sections start by formally defining the respective security properties (Subsections 3.1 and 4.1); a type system is then presented, and its soundness is proved (Subsections 3.2 and 4.2). Finally we discuss related work (Section 5) and conclude (Section 6). Due to space constraints, proofs are omitted from the paper, but appear in detail in the Appendix.

2. Language

The language that we consider for the study of local computations is a distributed imperative higher-order λ-calculus with reference and thread creation, where we include a flow policy declaration construct (for directly manipulating flow policies [1, 2]) and the new allowed flow policy tester construct that branches according to whether a certain flow policy is allowed in the program's computing context. We also add a notion of computation domain, to which we associate an allowed flow policy, and a code migration primitive. Programs computing 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. This will allow us to avoid synchronization issues that are not central to this work (and that are already handled elsewhere [1]). Nevertheless, as we will see in Subsection 2, this assumption does not imply the loss of the model's distributed nature, since the location in which programs compute will have a direct impact on the results we study here.

Syntax *Security annotations and types* are apparent in the syntax of the language, though they do not play any role in the operational semantics (they will be used at a later stage of the analysis). Security levels l, j, k are sets of principals, which are ranged over by $p, q \in$ ***Pri***. They are associated to references (and reference creators), representing the set of principals that are allowed to read the information contained in each reference. We also decorate references with the type of the values that they can hold. The syntax of types τ, σ, θ is given later in Subsections 3.2 and 4.2. In the following we may omit reference subscripts whenever they are not relevant. A security level is also associated to each thread, and appears as a subscript of thread names. This level can be understood as the set of principals that are allowed to know about the location of the thread in the network. Flow policies A, F, G are binary relations over ***Pri***. A pair $(p, q) \in F$, most often written $p \prec q$, is to be understood as "information may flow from principal p to principal q", that is, more precisely, "everything that principal p is allowed to read may also be read by principal q". We denote, as usual, by F^* the reflexive and transitive closure of F.

The language of *threads* (defined in Figure 1) is based on a call-by-value λ-calculus extended with the imperative constructs of ML, conditional branching and boolean values (here the $(\varrho x.W)$ construct provides for recursive values). Variables x and references a, b, c are drawn from two disjoint countable sets ***Var*** and ***Ref***, respectively. Reference names can be created at runtime. There are two new kinds of names, given to threads (m, n) and to domains (d), each drawn from two new disjoint countable sets ***Dom*** $\neq \emptyset$ and ***Nam***. The new features are the flow declaration and the allowed-condition. The flow declaration construct is written $(\mathrm{flow}\ F\ \mathrm{in}\ M)$, 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 $(\mathrm{allowed}\ F\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f)$ 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 thread creator $(\mathrm{thread}_l\ M)$ spawns a thread M, to which a name and the security level l is given, and returns (); the new thread is to be executed concurrently. The meaning of the migration construct $(\mathrm{goto}\ d)$, where d is a domain name is that the thread that executes the migration operation should migrate to the domain d.

Networks are flat juxtapositions of domains, each containing a store and a pool of threads, which are subjected to the flow policy of the domain. They are in fact just a collection of references, threads that are running in parallel, and the flow policies allowed by each domain. Threads run concurrently in *pools* $P : ($***Nam*** $\times 2^{\textbf{\textit{Pri}}}) \rightarrow$ ***Exp***, which are mappings from decorated thread names to expressions (they can also be seen as sets of threads). *Stores* $S : ($***Ref*** $\times 2^{\textbf{\textit{Pri}}} \times$ ***Typ***$) \rightarrow$ ***Val*** map decorated reference names to values. To keep track of the locations of threads it suffices to maintain a mapping from thread names to domain names. This is the purpose of the *position-tracker* $T : ($***Nam*** $\times 2^{\textbf{\textit{Pri}}}) \rightarrow$ ***Dom***, which is a mapping from a finite set of decorated thread names to domain names. The pool P containing all the threads in the network, the mapping 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. The flow policies that are allowed by each domain are kept by the *policy-mapping* $W :$ ***Dom*** $\rightarrow 2^{\textbf{\textit{Pri}} \times \textbf{\textit{Pri}}}$ from domain names to flow policies, which is considered fixed in this model.

Operational semantics We now define the semantics of the language as a small step operational semantics on configurations. The call-by-value evaluation order can be conveniently specified by writing expressions using *evaluation contexts*. Intuitively, expressions that are placed in such contexts are to be executed first. We write $\mathrm{E}[M]$ to denote an expression where the subexpression M is placed in the evaluation context E, obtained by replacing the occurrence of [] in E by M.

Variables	x, y	$\in$	***Var***	*Thread Names*	m, n	$\in$	***Nam***
Reference Names	a, b, c	$\in$	***Ref***	*Domain Names*	d	$\in$	***Dom***

Values	V	$\in$	***Val***	$::=$	$() \mid x \mid a_{l,\theta} \mid (\lambda x.M) \mid tt \mid f\!f$
Pseudo-values	W	$\in$	***Pse***	$::=$	$V \mid (\varrho x.W)$
Expressions	M, N	$\in$	***Exp***	$::=$	$W \mid (M\ N) \mid (M; N) \mid (\text{if } M \text{ then } N_t \text{ else } N_f) \mid (\text{ref}_{l,\theta}\ M) \mid (!\ N) \mid (M := N) \mid$ $(\text{thread}_l\ M) \mid (\text{goto } d) \mid \boldsymbol{(\textbf{flow } F \textbf{ in } M)} \mid \boldsymbol{(\textbf{allowed } F \textbf{ then } N_t \textbf{ else } N_f)}$
Threads				$::=$	$M^{m_j}\ \ (\in \textbf{\textit{Exp}} \times \textbf{\textit{Nam}} \times 2^{\textbf{\textit{Pri}}})$

Figure 1. Syntax of Threads

$$\textit{Evaluation Contexts}\quad \text{E} \ ::= \ [] \mid \boldsymbol{(\textbf{flow } F \textbf{ in E})} \mid$$
$$(\text{E}\ N) \mid (V\ \text{E}) \mid (\text{E}; N) \mid (\text{ref}_{l,\theta}\ \text{E}) \mid (!\ \text{E}) \mid$$
$$(\text{E} := N) \mid (V := \text{E}) \mid (\text{if E then } N_t \text{ else } N_f)$$

The analysis of whether the information flows that occur in M are to be allowed depends on the flow policies that are declared in the evaluation context where M is executed. We denote by $\lceil \text{E} \rceil$ the flow policy that is permitted by the evaluation context E. It collects all the flow policies that are declared using flow declarations into one single flow policy:

Definition 2.1 (Flow Policy Declared by an Evaluation Context)**.** *The* flow policy declared by the evaluation context E *is given by* $\lceil \text{E} \rceil$ *where:*

$$\lceil [] \rceil = \emptyset, \qquad \lceil (\text{flow } F \text{ in E}) \rceil = F \cup \lceil \text{E} \rceil,$$
$$\lceil \text{E}'[\text{E}] \rceil = \lceil \text{E} \rceil, \ \textit{if } \text{E}' \textit{ does not contain flow declarations}$$

Some basic notations and conventions are useful for defining transitions on configurations. Given a configuration $\langle P, T, S \rangle$, we call the pair $\langle T, S \rangle$ the *state* of the configurations. We define $\text{dom}(S)$ as the set of decorated reference names that are mapped by S; similarly, the sets $\text{dom}(W)$, $\text{dom}(P)$ and $\text{dom}(T)$, are the sets of domains and decorated names of threads that are mapped by W, P and T. We say that a thread or reference name is fresh in T or S if it does not occur, with any subscript, in $\text{dom}(T)$ or $\text{dom}(S)$, respectively. We denote by $\text{tn}(P)$ and $\text{rn}(P)$ the set of decorated thread and reference names, respectively, that occur in the expressions of P (this notation is extended in the obvious way to expressions). 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 following additional conditions for memories, values stored in memories, and thread names: $\text{rn}(P) \subseteq \text{dom}(S)$; for any $a_{l,\theta} \in \text{dom}(S)$ we have $\text{rn}(S(a_{l,\theta})) \subseteq \text{dom}(S)$ $\text{dom}(P) \subseteq \text{dom}(T)$; $\text{tn}(\text{dom}(S)) \subseteq \text{dom}(T)$; all threads in a configuration have distinct names, and also all occurrences of a name in a configuration are decorated in the same way. 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 transitions of our *small step semantics* are defined in Figure 2. In the first group of rules, corresponding to local computations, the '$A \vdash$' turnstile makes explicit the allowed flow policy A of the site where the computations are taking place. The semantics of local evaluation is embedded in the distributed language, by means of the last two rules in Figure 2. This is formalized in the former rule by specifying the local flow policy A as $W(T(m_j))$, where $T(m_j)$ represents the location of the thread m_j that is being considered. The last rule establishes that the execution of a pool of threads is compositional (up to the expected restriction on the choice of new names). The semantics of global computations introduces the rule for thread creation and the rule for migration, which depends on the type system of Section 4[1]. Detailed explanations on the meaning of the typing judgment in its side condition are postponed to Subsection 4.2. For now, it is enough to know that s represents an approximation of the flow policies that are used by the typed expression and are not protected (in the sense explained in the introduction) by an appropriate allowed-condition.

The labeled transition rules of our semantics are decorated with the flow policy declared by the evaluation context where they are performed. Most of the transitions do not depend on the flow label F that decorates them. In particular, the evaluation of $(\text{flow } F \text{ in } M)$ simply consists in the evaluation of M, annotated with a flow policy that comprises (in the sense of set inclusion) F. The lifespan of the flow declaration terminates when the expression M that is being evaluated terminates (that is, M becomes a value). The flow policy that decorates the transition steps is used only by the rules for $(\text{allowed } F \text{ then } N_t \text{ else } N_f)$, whose semantics is similar to the conditional branching, but where the choice of the branch depends on whether F is allowed to be declared or not.

The allowed flow policy A of a site represents a restriction on the flow policies that can be set up by programs

[1]A similar rule appears in [12]. The side condition represents the standard theoretical requirement of checking incoming code before allowing it to execute in a given machine. It abstracts away from the details of how the migration control is implemented.

$$\begin{array}{rcl}
A \vdash \langle \mathrm{E}[((\lambda x.M)\ V)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[\{x \mapsto V\}M], S\rangle \\
A \vdash \langle \mathrm{E}[(\text{if } tt \text{ then } N_t \text{ else } N_f)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[N_t], S\rangle \\
A \vdash \langle \mathrm{E}[(\text{if } f\!f \text{ then } N_t \text{ else } N_f)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[N_f], S\rangle \\
\boldsymbol{A \vdash \langle \mathbf{E}[(\textbf{allowed } F \textbf{ then } N_t \textbf{ else } N_f)], S\rangle} & \xrightarrow[\lceil \mathbf{E} \rceil]{} & \boldsymbol{\langle \mathbf{E}[N_t], S\rangle}, \ \textit{\textbf{where}}\ \boldsymbol{F \subseteq A^*} \\
\boldsymbol{A \vdash \langle \mathbf{E}[(\textbf{allowed } F \textbf{ then } N_t \textbf{ else } N_f)], S\rangle} & \xrightarrow[\lceil \mathbf{E} \rceil]{} & \boldsymbol{\langle \mathbf{E}[N_f], S\rangle}, \ \textit{\textbf{where}}\ \boldsymbol{F \not\subseteq A^*} \\
A \vdash \langle \mathrm{E}[(V; N)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[N], S\rangle \\
A \vdash \langle \mathrm{E}[(\varrho x.W)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[(\{x \mapsto (\varrho x.W)\}\ W)], S\rangle \\
A \vdash \langle \mathrm{E}[(\text{flow } F \text{ in } V)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[V], S\rangle \\
A \vdash \langle \mathrm{E}[(!\ a_{l,\theta})], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[V], S\rangle, \ \textit{where } S(a_{l,\theta}) = V \\
A \vdash \langle \mathrm{E}[(a_{l,\theta} := V)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[()], [a_{l,\theta} := V]S\rangle \\
A \vdash \langle \mathrm{E}[(\mathrm{ref}_{l,\theta}\ V)], S\rangle & \xrightarrow[\lceil \mathrm{E} \rceil]{} & \langle \mathrm{E}[a_{l,\theta}], [a_{l,\theta} := V]S\rangle, \ \textit{a fresh in } S
\end{array}$$

$$W \vdash \langle \{\mathrm{E}[(\mathrm{thread}_l\ N)]^{m_j}\}, T, S\rangle \xrightarrow[\lceil \mathrm{E} \rceil]{} \langle \{\mathrm{E}[()]^{m_j}, N^{n_k}\}, [n_k := T(m_j)]T, S\rangle, \ \textit{where } n \textit{ fresh in } T$$

$$\boldsymbol{W \vdash \langle \{\mathbf{E}[(\textbf{goto } d)]^{m_j}\}, T, S\rangle} \xrightarrow[\lceil \mathbf{E} \rceil]{} \boldsymbol{\langle \{\mathbf{E}[()]^{m_j}\}, [m_j := d]T, S\rangle}, \textit{\textbf{where}}\ \boldsymbol{\Gamma \vdash \mathbf{E}[()] : s, \tau} \ \textit{\textbf{and}}\ \boldsymbol{s \subseteq W(d)^*}$$

$$\frac{W(T(m_j)) \vdash \langle M, S\rangle \xrightarrow[F]{} \langle M', S'\rangle}{W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F]{} \langle \{M'^{m_j}\}, T', S'\rangle} \qquad \frac{W \vdash \langle P, T, S\rangle \xrightarrow[F]{} \langle P', T', S'\rangle \quad \langle P \cup Q, T, S\rangle \textit{ is well formed}}{W \vdash \langle P \cup Q, T, S\rangle \xrightarrow[F]{} \langle P' \cup Q, T', S'\rangle}$$

Figure 2. Operational Semantics. See Figure 4 for the side condition of the migration rule.

running in that site. At the level of the semantics, the site's allowed flow policy A is used to determine the behavior of the allowed-condition, which 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 be:

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

The allowed flow policy is also used at migration time, to determine whether or not a migration instruction may be consummated. The idea is that a thread can only migrate to another domain if it respects its allowed flow policy. E.g., as will become clear in Subsection 4.2, the configuration

$$\langle \{\mathrm{E}[((\text{goto } d); (\text{flow } F \text{ in } M))]^{m_j}\}, T, S\rangle \tag{2}$$

can only perform an execution step if $W(d)$ allows for F; otherwise it gets stuck. Notice that the flow declaration does not imply checks to the allowed flow policy of the site. Here we preserve the original semantics of the flow declaration [2] as a construct that does not change the behavior of programs. The functionality of inspecting the allowed flow policy is thus restricted to the allowed-condition[2].

Thread names are used in three situations: When a new thread is created, its fresh name is added to the position-tracker, associated to the parent domain. When the $(\text{goto } d)$ statement is executed by a thread m, the position of m in the position-tracker is updated to d. Thread names are also (implicitly) 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.

[2]Besides avoiding the computational cost of continuously checking the allowed flow policy of the current domain, this design decision avoids some redundancy at the level of the semantics design. Here we wish to separate the enabling vs. controlling dimensions of declassification, and leave it to the security analysis mechanism to match flow declarations against the allowed-conditions of the context where they appear.

Distribution According to the chosen semantics, dereferencing and assigning to a reference can be done by all threads, regardless of their position in the network. One may wonder whether it is reasonable to consider a system with a shared global state as distributed. We point out that in this model, by associating different allowed flow policies to different computation domains, where programs have the power to test the allowed flow policy of the site they are located at, the behavior of a program fragment may differ on different machines. As an example, the thread

$$(\text{allowed } F \text{ then } (y_L := 1) \text{ else } (y_L := 2))^{m_j} \tag{3}$$

running in a network $\langle P, T, S\rangle$ such that $W(d_1) = F_1$ and $W(d_2) = F_2$, where $F \subseteq F_2^*$ but $F \not\subseteq F_2^*$. The thread will perform different assignments depending on whether $T(m_j) = d_1$ or $T(m_j) = d_2$. In Subsection 3.1 we will see that their behavior is distinguishable by the information flow bisimulation relation we are interested in this paper. In other words, the network does exhibit a distributed behavior. For a study of a similar model with distributed and mobile references, see [1].

3. Information flow analysis

In this section we start by briefly defining the security property of Non-disclosure for Networks, the underlying information flow policy that this work is based on (we refer the reader to [1] for further explanations). We will see that a new form of migration leaks appears due to the new allowed-condition primitive that was introduced in our language. We then present a type system for enforcing non-disclosure, and state its soundness.

3.1. Non-Disclosure for Networks

The study of confidentiality traditionally relies on a lattice of security levels [10], corresponding to security clearances that can be associated to information containers in a programming language. Here, as in [2], we will use a more general structure, that of a pre-lattice (a preordered set such that any two elements have a least upper-bound and a greatest lower-bound), that is sufficient and convenient for defining a dynamic flow relation that accounts for runtime changes in the flow policy of a program. More concretely, our security pre-lattices are derived from a security lattice where security levels are sets of principals representing read-access rights, partially ordered by the reverse inclusion relation, which indicates allowed flows of information: if $l_1 \supseteq l_2$ then information in a reference a_{l_1} may be transferred to b_{l_2}, since the principals allowed to read this value from b were already allowed to read it from a. Flow policies, which are binary relations between principals, then represent additional directions in which information is allowed to flow. This leads to the underlying *preorder on security levels*, given by $l_1 \preceq_F l_2 \overset{\text{def}}{\Leftrightarrow} (l_1 \uparrow_F) \supseteq (l_2 \uparrow_F)$, where the *$F$-upward closure* of a security level l, defined as $l \uparrow_F = \{q \mid \exists p \in l.\ p\ F^*\ q\}$ contains all the principals that are allowed by the policy F to read the contents of a reference labeled l. We choose $l_1 \curlywedge_F l_2 = l_1 \cup l_2$ and $l_1 \curlyvee_F l_2 = (l_1 \uparrow_F) \cap (l_2 \uparrow_F)$ as meet and join operations, from which $\top = \emptyset$ and $\bot =$ ***Pri***. Notice that $\preceq_F$ extends $\supseteq$ in the sense that $\preceq_F$ is larger than $\supseteq$ and that $\preceq_\emptyset = \supseteq$.

Equipped with a flow relation between security levels, we can define the notions of low part of a state and of low-equality between states with respect to a flow policy F and security level l. Intuitively, two states are said to be "low-equal" if they have the same "low-domain", and if they give the same values to all objects (in this case, references and threads) that are labeled with "low" security levels.

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 is given by*

$$\langle T_1, S_1\rangle =^{F,l} \langle T_2, S_2\rangle \overset{\text{def}}{\Leftrightarrow} \begin{array}{l} T_1 \upharpoonright^{F,l} = T_2 \upharpoonright^{F,l} \text{ and} \\ S_1 \upharpoonright^{F,l} = S_2 \upharpoonright^{F,l} \end{array}$$

$$\text{where: } \begin{array}{l} T \upharpoonright^{F,l} \overset{\text{def}}{=} \{(n_k, d) \mid (n_k, d) \in T \ \&\ k \preceq_F l\} \\ S \upharpoonright^{F,l} \overset{\text{def}}{=} \{(a_{k,\theta}, V) \mid (a_{k,\theta}, V) \in S \ \&\ k \preceq_F l\} \end{array}$$

This relation is transitive, reflexive and symmetric.

Given that we are considering a concurrent (and distributed) setting, it is natural to formulate our information flow property in terms of a bisimulation [5, 24]. Our bisimulation, which is based on the small-step semantics defined in Section 2, relates two pools of threads if they show the same behavior on the low part of two states. In the following we denote by $\rightarrow^*$ the reflexive and transitive closure of the union of the transitions $\underset{F}{\longrightarrow}$, for all F.

Definition 3.2 ($\approx_l$)**.** *An l-bisimulation is a symmetric relation $\mathcal{R}$ on sets of threads such that, for all T_1, S_1, T_2, S_2:*

$P_1\ \mathcal{R}\ P_2$ and $W \vdash \langle P_1, T_1, S_1\rangle \underset{F}{\longrightarrow} \langle P_1', T_1', S_1'\rangle$ and
$\langle T_1, S_1\rangle =^{F,l} \langle T_2, S_2\rangle$ implies
$\exists P_2', T_2', S_2' .\ W \vdash \langle P_2, T_2, S_2\rangle \rightarrow^* \langle P_2', T_2', S_2'\rangle$ and
$\langle T_1', S_1'\rangle =^{\emptyset,l} \langle T_2', S_2'\rangle$ and $P_1'\ \mathcal{R}\ P_2'$ when

$(\mathrm{dom}(S_1{}') - \mathrm{dom}(S_1)) \cap \mathrm{dom}(S_2) = \emptyset$ *and:*
$(\mathrm{dom}(T_1{}') - \mathrm{dom}(T_1)) \cap \mathrm{dom}(T_2) = \emptyset$

The largest l-bisimulation[3] *is denoted by $\approx_l$.*

The above bisimulation potentially relates more programs than one for Non-interference thanks to the stronger premise $S_1 =^{F,l} S_2$. By starting with pairs of memories that are low-equal "to a greater extent", i.e. that coincide in a larger portion of the memory, the condition on the behavior of the program P_2 becomes weaker. The bisimulation relation could have been parameterized by an additional flow policy G, which would represent the global flow policy that is assumed to hold everywhere by default. In this paper we chose to omit this parameter by fixing $G = \emptyset$, for notational clarity, though all the results can be easily extended accordingly. For simplicity of the bisimulation definition, we are also not concerned with the fact that this definition can be considered somewhat restrictive in what respects changes in references that are created at run-time.

Note that the relation $\approx_l$ is not reflexive. For instance, the insecure expression $(v_B := (!\ u_A))$ is not bisimilar to itself if $A \not\preceq_F B$. In fact, if a program is shown to be bisimilar to itself, one can conclude that the high part of the state has not interfered with the low part, i.e., no security leak has occurred. This motivates the definition of our security property:

Definition 3.3 (Non-disclosure for Networks)**.** *A pool of threads P satisfies the Non-disclosure for Networks policy if it satisfies $P \approx_l P$ for all security levels l.*

[3]Note that for any l there is an l-bisimulation, like for instance the set of pairs of named values. Furthermore, the union of a family of l-bisimulations is an l-bisimulation, which is the largest l-bisimulation.

Intuitively, the above definition requires information flows occurring at any computation step that can be performed by some thread in a network, to comply with the flow policy that is declared by the context where the command is executed.

Migration leaks 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 on memory accesses [1]. However, in our setting, 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:

$$\begin{array}{l}(\text{if } (!\, x_H) \text{ then } (\text{goto } d_1) \text{ else } (\text{goto } d_2))\ ;\\ (\text{allowed } F \text{ then } (y_L := 1) \text{ else } (y_L := 2))\end{array} \qquad (4)$$

In this example, the thread will 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.

The fact that synchronization issues that are typical of distributed settings appear in spite of the state being globally shared allows us to make the point that migration leaks are not specific to distributed memory models. In fact, they can occur whenever the semantics or behavior of a program fragment differs on different machines.

3.2. Type System

We now present a type and effect system that accepts programs that satisfy Non-disclosure for Networks, as defined in Subsection 3.1. The judgments of the type and effect system, presented in Figure 3, have the form

$$\Gamma \vdash^j_F M : s, \tau$$

meaning that the expression M is typable with type τ and security effect s in the typing context Γ : ***Var*** $\rightarrow$ ***Typ***, which assigns types to variables. The turnstile has two parameters: 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; the security level j represents the confidentiality level associated to the thread that the expression M is part of, which is the confidentiality level of the position 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 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):

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

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 $\theta\ \mathrm{ref}_l$. Here the security level l is used to determine the effects of expressions that handle references. Typable expressions that reduce to a function that takes a parameter of type τ, that returns an expression of type σ, and with a *latent* [15] effect s, flow policy F and security level j have the function type $\tau \xrightarrow[F,j]{s} \sigma$. The latent effect is the security effect of the body of the function, while the latent flow policy is the one assumed to hold when the function is applied to an argument, and the latent security level j is that of the thread containing the expression.

We use a (join) pre-semilattice on security effects, that is obtained from the pointwise composition of the pre-lattices of the security effects. More precisely:

$$\begin{array}{c} s \preceq_F s' \overset{\text{def}}{\Leftrightarrow} s.r \preceq_F s'.r \ \&\ s'.w \preceq_F s.w\ \& s.t \preceq_F s'.t \\ s \curlyvee_F s' \overset{\text{def}}{\Leftrightarrow} \langle s.r \curlyvee_F s'.r, s.w \curlywedge_F s'.w, s.t \curlyvee_F s'.t \rangle \\ \bot = \langle \textbf{\textit{Pri}}, \emptyset, \textbf{\textit{Pri}} \rangle \end{array}$$

We use some abbreviations to alleviate the notation of the typing judgments and operations, namely we write $\Gamma \vdash M : \tau$ when $\Gamma \vdash^j_F M : \langle \bot, \top, \bot \rangle, \tau$, which is mainly used when typing (pseudo)-values, and we write $\boldsymbol{s \curlyvee s'}$ when $\boldsymbol{s \curlyvee_\emptyset s'}$, which is used when constructing the security effects of the typed expressions.

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 "$\preceq_F$" 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.

The 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) deserve further attention: the security level j that is associated to each thread,

$$[\textsc{Nil}]\ \Gamma \vdash () : \mathsf{unit} \qquad [\textsc{BoolT}]\ \Gamma \vdash tt : \mathsf{bool} \qquad [\textsc{BoolF}]\ \Gamma \vdash f\!f : \mathsf{bool} \qquad [\textsc{Loc}]\ \Gamma \vdash a_{l,\theta} : \theta\ \mathrm{ref}_l$$

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

$$[\textsc{Flow}]\ \frac{\Gamma \vdash^j_{F \cup F'} N : s, \tau}{\Gamma \vdash^j_F (\mathrm{flow}\ F'\ \mathrm{in}\ N) : s, \tau} \qquad [\textsc{Allow}]\ \frac{\begin{array}{c}\boldsymbol{\Gamma \vdash^j_F N_t : s_t, \tau} \\ \boldsymbol{\Gamma \vdash^j_F N_f : s_f, \tau}\end{array} \quad \boldsymbol{j \preceq_F s_t.w, s_f.w}}{\boldsymbol{\Gamma \vdash^j_F (\mathbf{allowed}\ F'\ \mathbf{then}\ N_t\ \mathbf{else}\ N_f) : s_t \curlyvee s_f \curlyvee \langle \bot, \top, j \rangle, \tau}}$$

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

$$[\textsc{Ass}]\ \frac{\Gamma \vdash^j_F M : s, \theta\ \mathrm{ref}_l \quad \Gamma \vdash^j_F N : s', \theta \quad \begin{array}{c} s.t \preceq_F s'.w \\ s.r, s'.r, s.t, s'.t \preceq_F l \end{array}}{\Gamma \vdash^j_F (M := N) : s \curlyvee s' \curlyvee \langle \bot, l, \bot \rangle, \mathsf{unit}}$$

$$[\textsc{Cond}]\ \frac{\Gamma \vdash^j_F M : s, \mathsf{bool} \quad \begin{array}{c}\Gamma \vdash^j_F N_t : s_t, \tau \\ \Gamma \vdash^j_F N_f : s_f, \tau\end{array} \quad s.r, s.t \preceq_F s_t.w, s_f.w}{\Gamma \vdash^j_F (\mathrm{if}\ M\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f) : s \curlyvee s_t \curlyvee s_f \curlyvee \langle \bot, \top, s.r \rangle, \tau} \qquad [\textsc{Seq}]\ \frac{\Gamma \vdash^j_F M : s, \tau \quad \Gamma \vdash^j_F N : s', \sigma \quad s.t \preceq_F s'.w}{\Gamma \vdash^j_F (M; N) : s \curlyvee s', \sigma}$$

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

$$[\textsc{Thr}]\ \frac{j \preceq_F l \quad \Gamma \vdash^l_\emptyset M : s, \mathsf{unit}}{\Gamma \vdash^j_F (\mathrm{thread}_l\ M) : \langle \bot, j \curlywedge s.w, \bot \rangle, \mathsf{unit}} \qquad [\textsc{Mig}]\ \Gamma \vdash^j_F (\mathrm{goto}\ d) : \langle \bot, j, \bot \rangle, \mathsf{unit}$$

Figure 3. Type and Effect System for Non-disclosure for Networks

and represents the confidentiality level of the position of the thread in the network is used to update the writing effect in the thread creation and migration rules, as well as the termination effect in the allowed-condition rule; on the other hand, it is constrained not to "precede low writes" in rule ALLOW, and to be a lower bound of runtime threads in rule THR. We refer the reader to [1] for further explanations on the remaining conditions.

One can prove a subject reduction result stating that the type of a thread is preserved by reduction, while its effects "weaken". When an expression executes a computation step, some of its effects may be performed by reading, updating or creating a reference or by creating or migrating a thread, while some may also be discarded when a branch in a conditional expression is taken. This result can then be used to verify that the the proposed type system ensures the non-disclosure property, i.e. that it constrains the usage of the new constructs introduced in this language in order to prevent them from encoding information leaks. In fact, security of expressions with respect to Non-disclosure is guaranteed by the type system:

Theorem 3.4 (Soundness for Non-disclosure for Networks.)**.** *Consider a pool of threads P. If for all $M^{m_j} \in P$ there exist Γ, s and τ such that $\Gamma \vdash^j_\emptyset M : s, \tau$, then P satisfies the Non-disclosure for Networks policy.*

Notice that 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. Confinement analysis

In this section we formally define Operational Confinement for Networks, a new security property that specifies the restricted usage of declassification instructions. We will present a simple type system for calculating the declassification effect of programs, which can be used by the semantics of the language to control migration between sites. We prove the soundness of the proposed migration control mechanism, and discuss other alternative approaches.

4.1. Operational Confinement

Here we will deal with relations between flow policies, which leads us to define a (meet) pre-semilattice of flow policies. We introduce the *preorder on flow policies* $\preceq$, thus overloading the notation for the flow relations on security levels. The meaning of relating two flow policies as in $F_1 \preceq F_2$ is that F_1 is more permissive than F_2, in the sense that F_1 encodes all the information flows that are enabled by F_2:

$$F_1 \preceq F_2 \overset{\text{def}}{\Leftrightarrow} F_2 \subseteq F_1^*$$

Where the meet operation is given by $F_1 \curlywedge F_2 = F_1 \cup F_2$. Consequently, we have $\top = \emptyset$.

We now define operational confinement with respect to an allowed flow policy, and justify the chosen formalization. The property is formulated abstractly for any distributed model that includes the concept of an allowed flow policy of a site, and whose semantics is decorated with the flow policies that are set up by each transition.

In light of the semantics of our flow declarations, we can predict which flow policies are declared by a program at runtime by observing the flow policy that decorates each of the possible steps it might take. Then, confinement to an allowed policy A means that every step is decorated with a flow policy F that is stricter than A:

Definition 4.1 (Operationally A-Confined Threads)**.** *Given a fixed policy-mapping W, a set $\mathcal{C}$ of threads is said to be a set of operationally A-confined threads if the following holds for any $M^{m_j} \in \mathcal{C}$, for all T, S:*

$W \vdash \langle\{M^{m_j}\}, T, S\rangle \underset{F}{\longrightarrow} \langle\{M'^{m_j}\}, T', S'\rangle$ *implies*
$A \preceq F$ *and* $M'^{m_j} \in \mathcal{C}$ *and*
$W \vdash \langle\{M^{m_j}\}, T, S\rangle \underset{F}{\longrightarrow} \langle\{M'^{m_j}, N'^{n_k}\}, T', S'\rangle$ *implies:*
$A \preceq F$ *and* $M'^{m_j}, N'^{n_k} \in \mathcal{C}$

We say that a thread M^{m_j} is operationally A-confined if it belongs to the largest set of operationally A-confined threads[4].

Operational A-confinement is useful from the point of view of a class of sites whose allowed flow policies are weaker than A. However, during global computations, the location of a program is not fixed, nor is the allowed flow policy that the program should comply to. This means that the notion of operational confinement to a single flow policy does not speak of compliance to the flow policies of all sites where each program might execute. Consider for instance

$$(\text{allowed } F \text{ then } (\text{flow } F \text{ in } M_1) \text{ else } M_2) \qquad (5)$$

where the flow declaration of the policy F is executed only if F has been tested as being allowed by the domain the program is located at. Assuming there are no flow declarations in M_1 and M_2, for every allowed flow policy A, this program is operationally A-confined. However, in a slight variation of the program where we introduce migration

$$(\text{allowed } F \text{ then } ((\text{goto } d); (\text{flow } F \text{ in } M_1)) \text{ else } M_2) \qquad (6)$$

the continuation of the program that can migrate to d is not operationally F-confined. It would then be convenient to formulate a more general notion of operational confinement that refers to the allowed flow policies of the sites where each part of the program actually executes.

[4]The largest of these sets exists, for analogous reasons to Footnote 3.

The following confinement property is set up on pairs that carry information about the location of each thread. The allowed flow policy of the current location of the thread is used to place a restriction on the flow policies that decorate the transitions, step-by-step.

Definition 4.2 (Operationally Confined Located Threads)**.** *Given a fixed policy-mapping W, a set $\mathcal{C}$ of pairs $\langle d, M^{m_j}\rangle$ is said to be a set of* operationally confined located threads *if the following holds for any $\langle d, M^{m_j}\rangle \in \mathcal{C}$, for all T, S:*

$W \vdash \langle\{M^{m_j}\}, T, S\rangle \underset{F}{\longrightarrow} \langle\{M'^{m_j}\}, T', S'\rangle$ *implies*
$W(T(m_j)) \preceq F$ *and* $\langle T'(m_j), M'^{m_j}\rangle \in \mathcal{C}$ *and*
$W \vdash \langle\{M^{m_j}\}, T, S\rangle \underset{F}{\longrightarrow} \langle\{M'^{m_j}, N'^{n_k}\}, T', S'\rangle$ *implies*
$W(T(m_j)) \preceq F$ *and* $\langle T'(m_j), M'^{m_j}\rangle, \langle T'(n_k), N'^{n_k}\rangle \in \mathcal{C}$
when:
$T(m_j) = d$

We say that a located thread $\langle d, M^{m_j}\rangle$ is operationally confined if it belongs to the largest set of operationally confined threads[5].

Intuitively, operational confinement means that for every execution step that is performed by a program at a certain site, the declared flow policy always complies to that sites allowed flow policy. We will return to Example 6 and its security analysis in Subsection 4.2, at the point where the semantics of migration can be fully understood.

From the definition of operational confinement of individual threads, we can derive a notion of network confinement by obtaining, from a pool of threads P and its corresponding position-tracker T, the set:

$$\text{pair}(P, T) \overset{\text{def}}{=} \{\langle d, M^{m_j}\rangle \mid M^{m_j} \in P \text{ and } T(m_j) = d\}$$

A network is said to be operationally confined if all of the pairs of threads and their location are operationally confined in the sense of Definition 4.2.

4.2. Type System

We now present a type and effect system that constructs a declassification effect that can be used to enforce Confinement (as defined in Subsection 4.1). The judgments of the type and effect system of Figure 4 are a lighter version of those that were used in Subsection 3.2. They have the form

$$\Gamma \vdash M : s, \tau$$

meaning that the expression M is typable with type τ and security effect s in the typing context Γ : ***Var*** $\rightarrow$ ***Typ***, which assigns types to variables.

[5]The largest of these sets exists, for analogous reasons to Footnote 3.

$$[\textsc{Flow}]\ \frac{\Gamma \vdash N : s, \tau}{\Gamma \vdash (\text{flow } F' \text{ in } N) : \boldsymbol{s \cup F'}, \tau} \qquad [\textsc{Allow}]\ \frac{\Gamma \vdash N_t : s_t, \tau \quad \Gamma \vdash N_f : s_f, \tau}{\Gamma \vdash (\text{allowed } F' \text{ then } N_t \text{ else } N_f) : \boldsymbol{s_t - F'} \curlywedge s_f, \tau}$$

Figure 4. Fragment of the Type and Effect System for Calculating the Declassification Effect

Here the security effect s corresponds to the *declassification effect*: a lower bound to the flow policies that are declared in the typed expression, excluding those that are positively tested by an allowed-condition. Types have the following syntax (t is a type variable):

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

The meanings are analogous to those of Subsection 3.2. We also use a similar abbreviation when typing (pseudo)-values, namely we write $\Gamma \vdash M : \tau$ when $\Gamma \vdash M : \top, \tau$.

In Figure 4 we only exhibit the typing rules that are relevant to the construction of the declassification effect. The omitted rules are a simplified version of those that appear in Figure 3, where typing judgments have no parameters, and the updates of the security effects as well as all side conditions involving $\preceq_F$ are removed. The meet operator $\curlywedge$ is used instead of $\curlyvee$ because the declassification effect is contravariant with respect to the effects of Subsection 3.2.

Our type and effect system accurately constructs the declassification effect of the program, which allows to test confinement of all flow declarations with respect to any allowed flow policy. The new effect is updated in rule FLOW, each time a flow declaration is performed, and "grows" as the declassification effects of subterms are met (by union) in order to form that of the parent command. An exception appears in rule ALLOW, where the declassification effect of the "allowed" branch is not used entirely: the part that has been tested by the allowed-condition is omitted. The intuition is that the part that is removed (say, F) is of no concern since in practice the allowed branch will only be executed if F is allowed.

As we have mentioned, the declassification effect should give information about the potential flow policy environments that are set up by the program. It is easy to see that the proposed type system provides a rather naive solution to this problem, since it does not take into consideration the migration instructions that appear in the code. This means that it might over-approximate the declassification effect by counting in flow declarations that are not relevant to the site where the program is arriving.

Here we are not concerned with the precision of the type system, but rather with putting forward its idea. As we will see ahead in Subsection 4.2, its imprecisions can be overcome by good programming practices, since rejection of programs can be avoided by proper usage of the allowed-condition construct. In spite of the simplicity of the type system, notice that it does take that effort into account when building the declassification effect. In fact, when a part of the program is "protected" by an $(\text{allowed } F \text{ then } M \text{ else } N)$ construct, some of the information in the declassification effect can be discarded. By hiding the tested flow policy from the declassification effect of the "allowed" branch. As a result, programs containing flow declarations that are too permissive might still be authorized to execute in a certain domain, as long as they occur in the "allowed" branch of our new construct, since as we know (by its semantics), it will never be executed.

Soundness We start by stating that the type of a thread is preserved by reduction, while its effects "weaken". When an expression executes a computation step, some of its effects may be performed by terminating computations within a flow declaration, or may also be discarded when a branch in a conditional expression is taken.

Proposition 4.3 (Subject Reduction)**.** *Consider a fixed policy-mapping W and a thread M^{m_j} for which there exist Γ, F, s and τ such that $\Gamma \vdash M : s, \tau$.*

- *If $W \vdash \langle\{M^{m_j}\}, T, S\rangle \xrightarrow[F']{} \langle\{M'^{m_j}\}, T', S'\rangle$, then there exists s' such that $\Gamma \vdash M' : s', \tau$, and where $s \cup W(T(m_j)) \preceq s'$.*
- *If $W \vdash \langle\{M^{m_j}\}, T, S\rangle \xrightarrow[F']{} \langle\{M'^{m_j}, N^{n_k}\}, T', S'\rangle$, then there exists s', s'' such that $\Gamma \vdash M' : s', \tau$ and $\Gamma \vdash N : s'', \mathsf{unit}$, and where $s \cup W(T(m_j)) \preceq s'$ and $s \cup W(T(n_k)) \preceq s''$.*

We will now verify that our migration control mechanism ensures the confinement property, i.e. that policies that are declared by flow declarations never violate the allowed flow policy of the domain where they are performed. To this end, we will see that the declassification effect can be used for enforcing operational confinement: if a program is typable with a declassification effect s, and if it performs an execution step in a context that enforces a flow policy F, then F is allowed by s, meaning that the actual flow policy that was ruling for that step is stricter than the declassification effect. To formalize this result, we use the notion of operational confinement with respect to a flow policy:

Proposition 4.4 (Meaning of the declassification effect)**.** *If for a thread M^{m_j} there exist Γ, G, s and τ such that $\Gamma \vdash M : s, \tau$, then M^{m_j} is operationally s-confined.*

According to the above result, a site can trust that declassifications performed by an incoming thread are not more permissive than what is declared in the type. Programs whose declassification effect cannot guarantee respect for the allowed flow policy of the site can then be treated as insecure by that site – in our model, they are simply forbidden to enter. This migration control mechanism allows us to formulate a network level soundness result, stating that typable programs can roam over the network with the guarantee that they will not violate the allowed flow policy of the sites where they execute:

Theorem 4.5 (Soundness of Typing Confinement for Networks)**.** *Consider a fixed policy-mapping W, a pool of threads P and its corresponding position tracker T, such that for all $M^{m_j} \in P$ there exist Γ, s and τ satisfying $\Gamma \vdash M : s, \tau$ and $W(T(m_j)) \preceq s$. Then the set $\text{pair}(P, T)$ is a set of operationally confined located threads.*

The above result might seem somewhat surprising at first, given that in the typing rule of the allowed construct, the flow policy that is being tested is subtracted from the declassification effect of the allowed branch. Notice however that the allowed branch will only be taken if the tested flow policy is allowed (by G) in the first place. This means that the part of the declassification effect of the allowed branch that is omitted is known to be allowed by G. To illustrate this idea, consider again Example 1, which has an empty declassification effect (and is therefore syntactically G-confined, for all G); its transitions can however be decorated with the flow policy $\{H \prec L\}$ in case the first branch is taken, which in turn can only happen if $\{H \prec L\} \subseteq G^*$.

Migration control The proposed method for enforcing operational confinement combines a static analysis technique for calculating the declassification effect of a program with a migration control technique that is built into the semantics of the language. Our analysis technique does not offer a safety result, guaranteeing that programs never "get stuck". In fact, it can happen that a thread is blocked at the point of performing a migration instruction, for the reason that it contains code that is not allowed to execute in the destination domain. Let us reconsider Example 6:

$$(\text{allowed}\, F \,\text{then}\, ((\text{goto}\, d); (\text{flow}\, F \,\text{in}\, M_1))\, \text{else}\, M_2)$$

According to the type system of Figure 4, the declassification effect of the continuation $((\text{goto}\, d); (\text{flow}\, F \,\text{in}\, M_1))$ includes F. This means that the migration instruction will be performed only in the case that the allowed flow policy of d allows for F; otherwise, the program will get stuck. We notice, however, that in order to avoid this situation, the program might have better been written

$$((\text{goto}\, d); (\text{allowed}\, F \,\text{then}\, (\text{flow}\, F \,\text{in}\, M_1)\, \text{else}\, M_2))$$

so that the flow declaration of F would not contribute to weaken the declassification effect of the continuation $(\text{allowed}\, F \,\text{then}\, (\text{flow}\, F \,\text{in}\, M_1)\, \text{else}\, M_2)$.

Here we follow the spirit of the proof carrying code scenario: The typing procedure provides enough information for a machine to decide whether the incoming program should be free to run in it or not. When using the type system to construct a declassification effect, we provide a way to build a certificate for the program, that can be analyzed at any time and place to conclude about whether a program should be allowed to execute or not. In case the certificate is not trusted, programs could also be statically checked to be "flow declaration safe".

5. Related work

Mantel and Sabelfeld [16] approach the study of information flow in a distributed setting by providing a type system for preserving confidentiality for different kinds of channels established over a publicly observable medium, but where interaction between domains is restricted to the exchange of values (no code mobility). Sharing our underlying aim of studying the distribution of code under decentralized security policies, Zdancewic *et al.* [27] study the secure partitioning of programs into a distributed system of potentially corrupted hosts and of principals. Crafa, Bugliesi and Castagna study non-interference for a purely functional distributed and mobile calculus [9], where no declassification mechanisms are contemplated. The work that is closest to ours is [1], which studies insecure information flows that are introduced by mobility in the context of a distributed language with states. Declassification is present in the language by means of flow declarations. In the computation model that is considered, threads own references that move along with them during migration; this setting also gives rise to *migration leaks*.

In a recent survey [23], Sabelfeld and Sands examine the literature regarding the subject of declassification, and observe that declassification can be controlled according to four main orthogonal goals as to: *what* information should be released [14, 21], *when* it should be allowed to happen [8], *who* should be authorized to use it [19], and *where* in the program it can be stated [2,7]; these dimensions can also be combined [3]. Most of the overviewed approaches implicitly assume local settings, where the computation platform enforces fixed policies. Furthermore, the tools that are provided for controlling the usage of declassification operations are restricted to the declassifying operations themselves, as opposed to the techniques that are proposed in this paper: both the allowed-construct and restricted version of migration are external to the flow declaration construct.

Previous works have studied forms of dynamic flow policy testing in settings where distribution and mobility are not explicitly dealt with. In [28] and [25], testing is performed over security labels, while the underlying security lattice remains fixed. Closer to ours is the work by Hicks *et al.* [13], where the global flow policy can be updated and tested. However, the language that is considered is local and sequential, and updates to the global flow policy are not meant as declassification operations. Furthermore, the security property does not deal with updates, but rather with what happens in between them. In [6] access control and declassification are combined in order to make sure that a program can only declassify information that it has the right to read, by using access control primitives for controlling the access level of programs that perform declassifications.

6. Conclusions and Future Work

The issue of controlling information flow in global computing is attracting increased attention Here we have motivated the need for controlling the usage of declassifying instructions in a global computing context by pointing out that 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. We have addressed this issue, by providing techniques for ensuring that a thread can only migrate to a site if it complies to its allowed flow policy. More concretely, the technical contributions of this paper are the following:

• A new programming construct $(\text{allowed}\, F\, \text{then}\, M\, \text{else}\, N)$ that tests the flexibility of the *allowed flow policy* imposed by the domain where it is currently located and can act accordingly. This is a first step towards programming in contexts where domains can change their allowed flow policies dynamically. In the presence of code mobility, the *allowed construct* provides useful expressibility for programming alternative behaviors that a program can have, should it end up in a site where certain declassification operations are not permitted.

• A new security property we call *confinement to a flow policy*, that ensures programs will respect certain flow policies, regardless of the declassification operations they might contain. This property is formulated at the network level, by considering distributed flow policies and the location of programs at runtime.

• A new form of security effect that can be associated to a program, containing information about the declassifying environments that can potentially be established by that program. We call it the *declassification effect*, and it is flexible enough to allow programs containing operations that are forbidden at certain sites to be considered secure nevertheless, as long as these operations are protected by an appropriate *allowed* construct. We show that the declassification effect can be easily constructed by means of a simple type and effect system. This information is useful when setting up a migration control mechanism for deciding whether or not programs should be allowed to execute at each site.

• The identification of a new form of *migration leaks* that can be encoded in a distributed language with code mobility by means of the new allowed construct. These do not result from memory synchronization issues, but reflect instead the new possibilities of accessing information about the location of programs in the network. We show how these migration leaks can be controlled by means of a type system that enforces the non-disclosure policy, for a distributed language with code mobility containing the allowed construct.

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 [1]. 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 [1]. 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.

Similarly to the Decentralized Label Model (DLM) [18], the (pre)-lattices of security levels that were considered here are based on the notion of principal, which can be seen as the underlying entity that holds read and write capabilities, and for which confidentiality is designed. The DLM model adds another dimension to it, by endowing principals with ownership capabilities. Another direction in which the model for attributing security labels to resources could be made more expressive would be by considering a connection between locations and principals.

The network model we studied in this paper assumes that the allowed flow policy of each domain is constant in time. It would be interesting to generalize the model in order to account for dynamic changes in these policies. However, combining this more general scenario with the allowed-condition would lead to inconsistencies, since the policies could potentially change after the branch of the allowed-condition had been chosen. This motivates the study of other alternatives to the allowed-condition for inspecting the *current* allowed flow policy of the context.

References

[1] A. Almeida Matos. *Typing Secure Information Flow: Declassification and Mobility*. PhD thesis, École Nationale Supérieure des Mines de Paris, 2006.

[2] A. Almeida Matos and G. Boudol. On declassification and the non-disclosure policy. In *CSFW'05: 18th IEEE Computer Security Foundations Workshop*, pages 226–240. IEEE Computer Society, 2005.

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

[4] G. Boudol. A generic membrane model. In C. Priami and P. Quaglia, editors, *GC'04: IST/FET International Workshop on Global Computing*, volume 3267 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2005.

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

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

[7] 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.

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

[9] S. Crafa, M. Bugliesi, and G. Castagna. Information flow security for boxed ambients. In V. Sassone, editor, *F-WAN'02: Workshop on Foundations of Wide Area Network Computing*, volume 66 of *Electronic Notes in Theoretical Computer Science*, pages 76–97. Elsevier, 2002.

[10] D. E. Denning. A lattice model of secure information flow. 19(5):236–243, 1976.

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

[12] 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.

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

[14] 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.

[15] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In *POPL'88: 15th ACM symposium on Principles of programming languages*, pages 47–57. ACM Press, 1988.

[16] 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.

[17] F. Martins and V. Vasconcelos. History-based access control for distributed processes. In *Proceedings of TGC'05*, LNCS. Springer-Verlag, 2005.

[18] 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.

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

[20] G. C. Necula. Proof-carrying code. In *POPL '97: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 106–119, New York, NY, USA, 1997. ACM.

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

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

[23] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *CSFW'05: 18th IEEE Computer Security Foundations Workshop*, pages 255–269. IEEE Computer Society, 2005.

[24] G. Smith. A new type system for secure information flow. In *CSFW'01: 14th IEEE Computer Security Foundations Workshop*, pages 115–125. IEEE Computer Society, 2001.

[25] S. Tse and S. Zdancewic. Run-time principals in information-flow type systems. In *IEEE 2004 Symposium on Security and Privacy*, pages 179–193. IEEE Computer Society Press, 2004.

[26] S. Zdancewic. Challenges for information-flow security. In *1st International Workshop on the Programming Language Interference and Dependence (PLID'04)*, 2004.

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

[28] L. Zheng and A. Myers. Dynamic security labels and noninterference. In *In Proc. 2nd Workshop on Formal Aspects in Security and Trust, IFIP TC1 WG1.7*, pages 27–40. Springer, 2004.

A. Notations

For notational convenience, in this appendix we represent transitions that refer to pools of single threads in a slightly different way. When a step is performed by a single thread while no new thread is created

$$W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F]{} \langle \{M'^{m_j}\}, T', S'\rangle$$

here we write:

$$W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F]{()} \langle \{M'^{m_j}\}, T', S'\rangle$$

When the step represents the creation of a new thread

$$W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F]{} \langle \{M'^{m_j}, N^{n_k}\}, T', S'\rangle$$

here we write

$$W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle \{M'^{m_j}\}, T', S'\rangle$$

Furthermore, we may represent all transitions of single threads by

$$W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle \{M'^{m_j}\}, T', S'\rangle$$

where N^{n_k} represents the dummy thread () in case no new thread is created.

B. Information flow analysis (proofs)

B.1. Subject Reduction

In order to establish the soundness of the type system of Figure 3 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 [?].

Remark B.1. *If $W \in$ **Pse** and $\Gamma \vdash^j_F W : s, \tau$, then for all flow policies F', we have that $\Gamma \vdash^j_{F'} W : \langle \bot, \top, \bot\rangle, \tau$.*

Lemma B.2.

1. *If $\Gamma \vdash^j_F M : s, \tau$ and $x \notin \mathrm{dom}(\Gamma)$ then $\Gamma, x : \sigma \vdash^j_F M : s, \tau$.*

2. *If $\Gamma, x : \sigma \vdash^j_F M : s, \tau$ and $x \notin \mathrm{fv}(M)$ then $\Gamma \vdash^j_F M : s, \tau$.*

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

Lemma B.3 (Substitution)**.**
If $\Gamma, x : \sigma \vdash^j_F M : s, \tau$ and $\Gamma \vdash W : \sigma$ then $\Gamma \vdash^j_F \{x \mapsto W\}M : s, \tau$.

Proof. By induction on the inference of $\Gamma, x : \tau \vdash^j_F M : s, \sigma$, and by case analysis on the last rule used in this typing proof, using the previous lemma. □

Lemma B.4 (Replacement)**.**
If $\Gamma \vdash^j_F \mathrm{E}[M] : s, \tau$ is a valid judgment, then the proof gives M a typing $\Gamma \vdash^j_{F \cup \lceil \mathrm{E} \rceil} M : \bar{s}, \bar{\tau}$ for some $\bar{s}$ and $\bar{\tau}$ such that $\bar{s}.r \preceq s.r$, $s.w \preceq \bar{s}.w$ and $\bar{s}.t \preceq s.t$. In this case, if $\Gamma \vdash^j_{F \cup \lceil \mathrm{E} \rceil} N : \bar{s}', \bar{\tau}$ with $\bar{s}'.r \preceq \bar{s}.r$, $\bar{s}.w \preceq \bar{s}'.w$ and $\bar{s}'.t \preceq \bar{s}.t$, then $\Gamma \vdash^j_F \mathrm{E}[N] : s', \tau$, for some s' such that $s'.r \preceq s.r$, $s.w \preceq s'.w$ and $s'.t \preceq s.t$.

Proof. By induction on the structure of E. □

Proposition B.5 (Subject Reduction)**.** *Consider a thread M^{m_j} for which there exist Γ, F, s and τ such that $\Gamma \vdash^j_F M : s, \tau$. Then, if $W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle \{M'^{m_j}\}, T', S'\rangle$, there exists s' such that $\Gamma \vdash^j_F M' : s', \tau$, and where $s'.r \preceq s.r$, $s.w \preceq s'.w$ and $s'.t \preceq s.t$. Furthermore, $\exists s''$ such that $\Gamma \vdash^k_\emptyset N : s''$,* unit *and $s.w \preceq s''.w$.*

Proof. We consider the smallest $\bar{M}$ such that $M = \bar{\mathrm{E}}[\bar{M}]$ 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 $W \vdash \langle \{\hat{M}^{m_j}\}, T, S\rangle \xrightarrow[\hat{F}]{\hat{N}^{n_k}} \langle \{\hat{M}'^{m_j}\}, T', S'\rangle$. We then proceed by case analysis on the transition $W \vdash \langle \{\bar{M}^{m_j}\}, T, S\rangle \xrightarrow[\bar{F}]{\bar{N}^{n_k}} \langle \{\bar{M}'^{m_j}\}, T', S'\rangle$, using Lemmas B.3 and B.4.

Suppose that $M = \bar{\mathrm{E}}[\bar{M}]$ and that $W \vdash \langle \{\bar{M}^{m_j}\}, T, S\rangle \xrightarrow[\bar{F}]{\bar{N}^{n_k}} \langle \{\bar{M}'^{m_j}\}, T', \bar{S}'\rangle$. We start by observing that this implies $F' = \bar{F} \cup \lceil \bar{\mathrm{E}} \rceil$, $M' = \bar{\mathrm{E}}[\bar{M}']$, $\bar{N} = N$ 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 $W \vdash \langle \{\hat{M}^{m_j}\}, T, S\rangle \xrightarrow[\hat{F}]{\hat{N}^{n_k}} \langle \{\hat{M}'^{m_j}\}, T', S'\rangle$.

By Lemma B.4, we have $\Gamma \vdash^j_{F \cup \lceil \bar{\mathrm{E}} \rceil} \bar{M} : \bar{s}, \bar{\tau}$ in the proof of $\Gamma \vdash^j_F \bar{\mathrm{E}}[\bar{M}] : s, \tau$, for some $\bar{s}$ and $\bar{\tau}$. We proceed by case analysis on the transition $W \vdash \langle \{\bar{M}^{m_j}\}, T, S\rangle \xrightarrow[\bar{F}]{\bar{N}^{n_k}} \langle \{\bar{M}'^{m_j}\}, T', S'\rangle$, and prove that $\Gamma \vdash^j_{F \cup \lceil \bar{\mathrm{E}} \rceil} \bar{M}' : \bar{s}', \bar{\tau}$, for some $\bar{s}'$ such that $\bar{s}'.r \preceq \bar{s}.r$, $\bar{s}.w \preceq \bar{s}'.w$ and $\bar{s}'.t \preceq \bar{s}.t$.

- **[$\bar{M} = ((\lambda x.\hat{M})\ V)$.]**
  Here we have $\bar{M}' = \{x \mapsto V\}\hat{M}$. By rule App, we have $\Gamma \vdash^j_{F \cup \lceil \bar{\mathrm{E}} \rceil} (\lambda x.\hat{M}) : \hat{s}, \hat{\tau} \xrightarrow[F \cup \lceil \bar{\mathrm{E}} \rceil, j]{\hat{s}'} \hat{\sigma}$ and $\Gamma \vdash^j_{F \cup \lceil \bar{\mathrm{E}} \rceil} V : \hat{s}'', \hat{\tau}$, where $\hat{s}'.r \preceq \bar{s}.r$, $\bar{s}.w \preceq \hat{s}'.w$ and $\hat{s}'.t \preceq \bar{s}.t$. By Abs, then $\Gamma, x : \hat{\tau} \vdash^j_{F \cup \lceil \bar{\mathrm{E}} \rceil} \hat{M} : \hat{s}', \hat{\sigma}$, and by Remark B.1 we have $\Gamma \vdash V : \hat{\tau}$. Therefore, by Lemma B.3, we get $\Gamma \vdash^j_{F \cup \lceil \bar{\mathrm{E}} \rceil} \{x \mapsto V\}\hat{M} : \hat{s}', \hat{\sigma}$.

- $[\bar{M} = (\text{if } tt \text{ then } N_t \text{ else } N_f).]$

  Here we have $\bar{M}' = N_t$. By COND, we have that $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} N_t : s_t, \bar{\tau}$, where $s_t.r \preceq \bar{s}.r$, $\bar{s}.w \preceq s_t.w$ and $s_t.t \preceq \bar{s}.t$.

- $[\bar{M} = (\text{ref}_{l,\theta}\ V).]$

  Here we have $\bar{M}' = a_{l,\theta}$. By LOC, we have $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} a : \langle\bot,\top,\bot\rangle, \theta\ \text{ref}_l$, which satisfies $\bot \preceq s.r$, $s.w \preceq \top$ and $\bot \preceq s.t$.

- $[\bar{M} = (!\ a_{l,\theta}).]$

  Here we have $\bar{M}' = S(a_{l,\theta})$. By assumption, we have that $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} S(a_{l,\theta}) : \langle\bot,\top,\bot\rangle, \theta$, which satisfies $\bot \preceq s.r$, $s.w \preceq \top$ and $\bot \preceq s.t$.

- $[\bar{M} = (\text{flow } F' \text{ in } V).]$

  Here we have $\bar{M}' = V$. By rule FLOW, we have that $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil\cup F'} V : \hat{s}', \tau$ and by Remark B.1, we have $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} V : \langle\bot,\top,\bot\rangle, \bar{\tau}$, which satisfies $\bot \preceq s.r$, $s.w \preceq \top$ and $\bot \preceq s.t$.

- $[\bar{M} = (\text{allowed } F' \text{ then } N_t \text{ else } N_f)$ and $F' \subseteq W(T(m_j))^*.]$

  Here we have $\bar{M}' = N_t$. By ALLOW, we have that $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} N_t : s_t, \bar{\tau}$, where $s_t.r \preceq \bar{s}.r$, $\bar{s}.w \preceq s_t.w$ and $s_t.t \preceq \bar{s}.t$.

- $[\bar{M} = (\text{allowed } F' \text{ then } N_t \text{ else } N_f)$ and $F' \not\subseteq W(T(m_j))^*.]$

  Here we have $\bar{M}' = N_f$. By ALLOW, we have that $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} N_f : s_f, \bar{\tau}$, where $s_f.r \preceq \bar{s}.r$, $\bar{s}.w \preceq s_f.w$ and $s_f.t \preceq \bar{s}.t$.

The proof for the case $\bar{M} = (\varrho x.W)$ is analogous to the one for $\bar{M} = ((\lambda x.\hat{M})\ V)$, while the proofs for the cases $\bar{M} = (\text{if } f\!f \text{ then } N_t \text{ else } N_f)$ and $\bar{M} = (V; \hat{M})$ are analogous to the one for $\bar{M} = (\text{if } tt \text{ then } N_t \text{ else } N_f)$, and the ones for $\bar{M} = (a_{l,\theta} := V)$, $\bar{M} = (\text{thread}_l\ \hat{M})$ is analogous to the one for $\bar{M} = (\text{ref}_{l,\theta}\ V)$. By Lemma B.4, we can conclude that $\Gamma \vdash^j_F \bar{\mathrm{E}}[\bar{M}'] : s', \tau$, for some s' such that $s'.r \preceq s.r$, $s.w \preceq s'.w$ and $s'.t \preceq s.t$.

Now, if $N^{n_k} \neq ()$ (N^{n_k} is created), then $\exists \hat{N} : M = \bar{\mathrm{E}}[(\text{thread}_k\ \hat{N})]$ and $\bar{N} = \hat{N}$. By Lemma B.4, we have $\Gamma \vdash^j_{F\cup\lceil\bar{\mathrm{E}}\rceil} (\text{thread}_k\ \hat{N}) : \hat{s}, \mathsf{unit}$ in the proof of $\Gamma \vdash^j_F \bar{\mathrm{E}}[(\text{thread}\ \hat{N})] : s, \tau$, for some $\hat{s}$, and $\hat{\tau}$. By THR, we have $\Gamma \vdash^k_\emptyset \hat{N} : \hat{s}, \mathsf{unit}$, where $\hat{s} = \langle\bot, s.w, \bot\rangle$. Therefore, $\hat{s}.r \preceq s.r$, $s.w \preceq \hat{s}.w$ and $\hat{s}.t \preceq s.t$. □

B.2. Non-disclosure for Networks

We now present the main steps for proving soundness of the type system of Figure 3 with respect to the notion of security of Definition 3.4.

B.2.1 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_j} differs in the context of two distinct states while not creating two distinct reference names or thread names, this is because either M^{m_j} is performing a dereferencing operation, which yields different results depending on the memory, or because M^{m_j} is testing the allowed policy.

Lemma B.6 (Splitting Computations)**.**
If we have that $W \vdash \langle\{M^{m_j}\}, T_1, S_1\rangle \xrightarrow[F]{N^{n_k}} \langle\{{M_1}'^{m_j}\}, T_1', S_1'\rangle$ *and also that* $W \vdash \langle\{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N'^{n_k}} \langle\{{M_2}'^{m_j}\}, T_2', S_2'\rangle$ *with* ${M_1}'^{m_j} \neq {M_2}'^{m_j}$ *and* $\text{dom}(T_2' - T_2) = \text{dom}(T_1' - T_1)$, $\text{dom}(S_2' - S_2) = \text{dom}(S_1' - S_1)$, *then* $N^{n_k} = () = N'^{n_k}$ *and either:*

- *there exist* E *and* $a_{l,\theta}$ *such that* $F = \lceil\mathrm{E}\rceil = F'$, $M = \mathrm{E}[(!\ a_{l,\theta})]$, *and* $M' = \mathrm{E}[S_1(a_{l,\theta})]$, $M'' = \mathrm{E}[S_2(a_{l,\theta})]$ *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*

- *there exist* E *and* $\bar{F}$ *such that* $F = \lceil\mathrm{E}\rceil = \bar{F}$, $M = \mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$, *and* $T_1(m_j) \neq T_2(m_j)$ *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.*

Proof. Note that the only rules that depend on the state are those for the reduction of $\mathrm{E}[(!\ a_{l,\theta})]$ and of $\mathrm{E}[(\text{allowed } F' \text{ then } N_t \text{ else } N_f)]$. By case analysis on the transition $W \vdash \langle\{M^{m_j}\}, T_1, S_1\rangle \xrightarrow[F]{N^{n_k}} \langle\{{M_1}'^{m_j}\}, T_1', S_1'\rangle$. □

Effects

Lemma B.7 (Update of Effects)**.**

1. *If* $\Gamma \vdash^j_F \mathrm{E}[(!\ n_k.u_{l,\theta})] : s, \tau$ *then* $l \preceq s.r$.

2. *If* $\Gamma \vdash^j_F \mathrm{E}[(n_k.u_{l,\theta} := V)] : s, \tau$, *then* $s.w \preceq l$.

3. *If* $\Gamma \vdash^j_F \mathrm{E}[(\text{ref}_{l,\theta}\ V)] : s, \tau$, *then* $s.w \preceq l$.

4. *If* $\Gamma \vdash^j_F \mathrm{E}[(\text{goto}\ d)] : s, \tau$, *then* $s.w \preceq j$.

5. *If* $\Gamma \vdash^j_F \mathrm{E}[(\text{allowed } F \text{ then } N_t \text{ else } N_f)] : s, \tau$, *then* $j \preceq s.t$.

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[6]:

Definition B.8 (Operationally High Threads)**.** *A set $\mathcal{H}$ of threads is said to be a set of* operationally (F, l)-high threads *if the following holds for any $M^{m_j} \in \mathcal{H}$:*

$$W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle \{M'^{m_j}\}, T', S'\rangle \text{ implies}$$

$$\langle T, S\rangle =^{F,l} \langle T', S'\rangle \text{ and both } M'^{m_j}, N^{n_k} \in \mathcal{H}$$

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

Remark that for any F and l there exists a set of operationally (F, l)-high threads, like for instance $\{V^{m_j} \mid V \in \textbf{\textit{Val}}\}$. 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 (F, l)-high thread is also operationally (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 B.9 (Syntactically High Expressions)**.** *An expression M is* syntactically (F, l, j)-high *if there exists Γ, A, s, τ such that $\Gamma \vdash^j_F M : s, \tau$ with $s.w \not\preceq_F l$. The expression M is a* syntactically (F, l, j)-high function *if there exists Γ, s, τ such that $\Gamma \vdash M : \tau \xrightarrow[F,j]{s} \sigma$ with $s.w \not\preceq_F l$.*

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

Lemma B.10 (High Expressions)**.** *If M is a syntactically (F, l, j)-high expression, then M^{m_j} is an operationally (F, l)-high thread.*

Proof. We show that if M is syntactically (F, l, j)-high, that is if there exists Γ, s, τ such that $\Gamma \vdash^j_F M : s, \tau$ with $s.w \not\preceq_F l$, and $W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle \{M'^{m_j}\}, T', S'\rangle$ then $S' =^{F,l} S$. This is enough since, by Subject Reduction (Theorem B.5), both M' is syntactically (F, l, j)-high and N is syntactically (F, l, k)-high. We proceed by cases on the proof of the transition $W \vdash \langle \{M^{m_j}\}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle \{M'^{m_j}\}, T', S'\rangle$. The lemma is trivial in all the cases where $\langle T, S\rangle = \langle T', S'\rangle$.

- **[$\boldsymbol{M = \mathrm{E}[(a_{\bar{l},\bar{\theta}} := V)]}$].** Here $S' = [a_{\bar{l},\bar{\theta}} := V]S$ and so $s.w \preceq \bar{l}$ by Lemma B.7. This implies $\bar{l} \not\preceq_F l$, hence $S' =^{F,l} S$.
- **[$\boldsymbol{M = \mathrm{E}[(\textbf{goto}\ d)]}$].** Here $T' = [m_j := d]T$ and also $s.w \preceq j$ by Lemma B.7. This implies $j \not\preceq_F l$, hence $T' =^{F,l} T$.

The proof of the case $M = \mathrm{E}[(\mathrm{ref}_{l,\theta}\ V)]$ is analogous to the proof for $M = \mathrm{E}[(a_{l,\theta} := V)]$, while the proof for the case $M = \mathrm{E}[(\mathrm{thread}_l\ M_0)]$ is analogous to the one for $M = \mathrm{E}[(\mathrm{goto}\ d)]$. ☐

B.2.2 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 ability of a thread containing a migrating instruction to compute depends on whether it is typable with a declassification effect that complies to the allowed flow policy of the destination site. The following guaranteed-transition result holds for low-equal states.

Lemma B.11 (Guaranteed Transitions)**.** *Consider a thread M^{m_j} that is typable for F. If $W \vdash \langle \{M^{m_j}\}, T_1, S_1\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle \{M_1'^{m_j}\}, T_1', S_1'\rangle$ such that $\bar{n}_{\bar{k}}$ is fresh for T_2 if $\bar{n}_{\bar{k}} \in \mathrm{dom}(T_1' - T_1)$ and a is fresh for S_2 if $a_{l,\theta} \in \mathrm{dom}(S_1' - S_1)$ and for some F' we have $\langle T_1, S_1\rangle =^{F \cup F', low} \langle T_2, S_2\rangle$, then there exist M_2', T_2' and S_2' such that $W \vdash \langle \{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle \{M_2'^{m_j}\}, T_2', S_2'\rangle$ with $\langle T_1', S_1'\rangle =^{F \cup F', low} \langle T_2', S_2'\rangle$.*

Proof. By case analysis on the proof of $W \vdash \langle \{M^{m_j}\}, T_1, S_1\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle \{M_1'^{m_j}\}, T_1', S_1'\rangle$. In most cases, this transition does not modify or depend on the state $\langle T_1, S_1\rangle$, and we may let $M_2' = M_1'$ and $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$.

- **[$\boldsymbol{M = \mathrm{E}[(\mathbf{ref}_{l,\theta}\ V)]}$].** Here $M' = \mathrm{E}[a_{l,\theta}]$, $F = \lceil \mathrm{E} \rceil$, $N^{\bar{n}_{\bar{k}}} = ()$, $T_1' = T_1$ and $S_1' = S_1 \cup \{a_{l,\theta} \mapsto V\}$. Since a is fresh for S_2,

[6]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 also have that $W \vdash \langle\{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle\{M_1'^{m_j}\}, T_2, S_2' \cup \{a_{l,\theta} \mapsto V\}\rangle$.

- $[\boldsymbol{M = \mathrm{E}[(!\, a_{l,\theta})]}.]$ Here $M' = \mathrm{E}[S_1(a_{l,\theta})]$, $F = \lceil \mathrm{E} \rceil$, $N^{\bar{n}_{\bar{k}}} = ()$, and $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$. We have $W \vdash \langle\{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle\{\mathrm{E}[S_2(a_{l,\theta})]^{m_j}\}, T_2, S_2\rangle$.

- $[\boldsymbol{M = \mathrm{E}[(a_{l,\theta} := V)]}.]$ then $M' = \mathrm{E}[()]$, $F = \lceil \mathrm{E} \rceil$, $N^{\bar{n}_{\bar{k}}} = ()$, $T_1' = T_1$ and $S_1' = [a_{l,\theta} := V]S_1$. We have $W \vdash \langle\{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle\{\mathrm{E}[()]^{m_j}\}, T_2, [a_{l,\theta} := V]S_2\rangle$.

- $[\boldsymbol{M = \mathrm{E}[(\mathrm{thread}_{\bar{k}}\, \bar{M})]}.]$ Here $M' = \mathrm{E}[()]$, $F = \emptyset$, $N^{\bar{n}_{\bar{k}}} = \bar{M}^{\bar{n}_{\bar{k}}}$, $T_1' = T_1 \cup \{\bar{n}_{\bar{k}} \mapsto T_1(m_j)\}$, and $S_1' = S_1$. Since n is fresh for T_2, $W \vdash \langle\{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle\{\mathrm{E}[()]^{m_j}\}, T_2 \cup \{\bar{n}_{\bar{k}} \mapsto T_2(m_j)\}, S_2\rangle$. Notice that $T_1 \cup \{\bar{n}_{\bar{k}} \mapsto T_1(m_j)\} =^{F \cup F', low} T_2 \cup \{\bar{n}_{\bar{k}} \mapsto T_2(m_j)\}$, because $T_1 =^{F \cup F', low} T_2$ and if $l \preceq_{F \cup F'} low$, then by the condition $j \preceq_{F \cup F'} l$ in rule THR also $j \preceq_{F \cup F'} low$, in which case $T_1(m_j) = T_2(m_j)$.

- $[\boldsymbol{M = \mathrm{E}[(\mathrm{goto}\, d')]}.]$ Then $M' = \mathrm{E}[()]$, $F = \emptyset$, $N^{\bar{n}_{\bar{k}}} = ()$, $T_1' = [m_j := d']T_1$ and $S_1' = S_1$. This means that $\Gamma \vdash^j_\emptyset \mathrm{E}[()] : s, \tau$. Therefore, we also have $W \vdash \langle\{M^{m_j}\}, T_2, S_2\rangle \xrightarrow[F]{N^{\bar{n}_{\bar{k}}}} \langle\{\mathrm{E}[()]^{m_j}\}, [m_j := d']T_2, S_2\rangle$.

□

We aim at proving that any typable thread M^{m_j} that has a low-termination effect always presents the same behavior according to a *strong* bisimulation on low-equal states: if two continuations $M_1^{m_j}$ and $M_2^{m_j}$ of M^{m_j} are related, and if $M_1^{m_j}$ can perform an execution step over a certain state, then $M_2^{m_j}$ 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_j} 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 B.11).

The definition of $\mathcal{T}^j_{F,low}$ is given in Figure 5. Notice that it is a symmetric relation. In order to ensure that expressions that are related by $\mathcal{T}^j_{F,low}$ 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.

Definition B.12 ($\mathcal{T}^j_{F,low}$). *We have that $M_1\ \mathcal{T}^j_{F,low}\ M_2$ if $\Gamma \vdash^j_F M_1 : s_1, \tau$ and $\Gamma \vdash^j_F M_2 : s_2, \tau$ for some Γ, A, s_1, s_2 and τ with $s_1.t \preceq_F low$ and $s_2.t \preceq_F low$ 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 $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, or*
- *[Clause 4.] $M_1 = (\mathrm{ref}_{l,\theta}\ \bar{M}_1)$ and $M_2 = (\mathrm{ref}_{l,\theta}\ \bar{M}_2)$ where $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $l \not\preceq_F low$, or*
- *[Clause 5.] $M_1 = (!\ \bar{M}_1)$ and $M_2 = (!\ \bar{M}_2)$ where $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, or*
- *[Clause 6.] $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ with $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $\bar{N}_1\ \mathcal{T}^j_{F,low}\ \bar{N}_2$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta\ \mathrm{ref}_l$ for some θ and l such that $l \not\preceq_F low$, or*
- *[Clause 7.] $M_1 = (\mathrm{flow}\ F'\ \mathrm{in}\ \bar{M}_1)$ and $M_2 = (\mathrm{flow}\ F'\ \mathrm{in}\ \bar{M}_2)$ with $\bar{M}_1\ \mathcal{T}^j_{F \cup F',low}\ \bar{M}_2$.*

Figure 5. The relation $\mathcal{T}^j_{F,low}$

Remark B.13. *If for some j, F and low we have $M_1\ \mathcal{T}^{j}_{F,low}\ M_2$ and $M_1 \in$ **Val**, then $M_2 \in$ **Val**.*

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

Lemma B.14. *If there exist Γ, s, τ such that $\Gamma \vdash^{j}_{F} \mathrm{E}[(!\ a_{l,\theta})] : s, \tau$ with $s.t \preceq_F low$ and $l \not\preceq_{F\cup\lceil \mathrm{E}\rceil} low$, then for any values $V_0, V_1 \in$ **Val** such that $\Gamma \vdash V_i : \theta$ we have $\mathrm{E}[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}[V_1]$.*

Proof. By induction on the structure of E.

- $[\mathbf{E}[(!\ a_{l,\theta})] = (!\ a_{l,\theta})$.] We have $V_0\ \mathcal{T}^{j}_{F,low}\ V_1$ by Clause 1.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{E_1}[(!\ a_{l,\theta})]\ M)$.] By App we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau} \xrightarrow[F,j]{\bar{s}'} \bar{\sigma}$ with $\bar{s}.r \preceq s.t$. By Lemma B.7, we have $l \preceq \bar{s}.r$. Therefore $l \preceq_F s.t$, which contradicts the assumption that both $s.t \preceq_F low$ and $l \not\preceq_{F\cup\mathrm{E}} low$ hold.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (V\ \mathbf{E_1}[(!\ a_{l,\theta})])$.] By rule App we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}'', \bar{\tau}$ with $\bar{s}''.r \preceq s.t$. By Lemma B.7, we have $l \preceq \bar{s}''.r$. Therefore $l \preceq_F s.t$, which contradicts the assumption that both $s.t \preceq_F low$ and $l \not\preceq_{F\cup\mathrm{E}} low$ hold.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{if}\ \mathbf{E_1}[(!\ a_{l,\theta})]\ \mathbf{then}\ M_t\ \mathbf{else}\ M_f)$.] By Cond we have that $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \mathsf{bool}$ with $\bar{s}.r \preceq s.t$. By Lemma B.7, we have $l \preceq \bar{s}.r$. Therefore $l \preceq_F s.t$, which contradicts the assumption that both $s.t \preceq_F low$ and $l \not\preceq_{F\cup\mathrm{E}} low$ hold.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{E_1}[(!\ a_{l,\theta})]; M)$.] By Seq we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau}$ with $\bar{s}.t \preceq_F s.t$. Therefore $\bar{s}.t \preceq_F low$, and since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. By Lemma B.4 and Clause 3 we can conclude.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{ref}_{l',\theta'}\ \mathbf{E_1}[(!\ a_{l,\theta})])$.] By rule Ref we have that $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau}$ with $\bar{s}.r = s.r \preceq_F l'$ and $\bar{s}.t = s.t$. Therefore $\bar{s}.t \preceq_F low$, and since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. By Lemma B.7 we have $l \preceq s.r$, so $s.r \not\preceq_F low$. Therefore, $l' \not\preceq_F low$, and we conclude by Lemma B.4 and Clause 4.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (!\ \mathbf{E_1}[a_{l,\theta}])$.] By rule Der we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau}$ with $\bar{s}.t \preceq_F s.t$. Therefore $\bar{s}.t \preceq_F low$, and since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. We conclude by Lemma B.4 and Clause 5.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{E_1}[(!\ a_{l,\theta})] := M)$.] By rule Ass we have that $\Gamma \vdash^{j}_{F} \mathrm{E}_1[a_{l,\theta}] : \bar{s}, \bar{\theta}\ \mathrm{ref}_{\bar{l},\bar{n}_k}$ with $\bar{s}.t \preceq_F s.t$ and $\bar{s}.r \preceq_F \bar{l}$. Therefore $\bar{s}.t \preceq_F low$, and since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. On the other hand, by Clause 2 we have $M\ \mathcal{T}^{j}_{F,low}\ M$. By Lemma B.7 we have $l \preceq \bar{s}.r$, so $l \preceq_F \bar{l}$. Then, we must have $\bar{l} \not\preceq_F low$, since otherwise $l \preceq_{F\cup\mathrm{E}} low$. Therefore, we conclude by Lemma B.4 and Clause 6.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (V := \mathbf{E_1}[(!\ a_{l,\theta})])$.] By rule Ass we have that $\Gamma \vdash^{j}_{F} V : \bar{s}, \bar{\theta}\ \mathrm{ref}_{\bar{l},\bar{n}_k}$, and $\Gamma \vdash^{j}_{F} \mathrm{E}_1[a_{l,\theta}] : \bar{s}', \theta$ with $\bar{s}'.t \preceq_F s.t$ and $\bar{s}'.r \preceq_F \bar{l}$. Therefore $\bar{s}'.t \preceq_F low$, and since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. On the other hand, by Clause 2 we have $V\ \mathcal{T}^{j}_{F,low}\ V$. By Lemma B.7 we have $l \preceq \bar{s}'.r$, so $l \preceq_F \bar{l}$. Then, we must have $\bar{l} \not\preceq_F low$, since otherwise $l \preceq_{F\cup\mathrm{E}} low$. We then conclude by Lemma B.4 and Clause 6.
- $[\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{flow}\ F'\ \mathbf{in}\ \mathbf{E_1}[(!\ a_{l,\theta})])$.] By rule Flow we have $\Gamma \vdash^{j}_{F\cup F'} V : s, \tau$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F\cup F',low}\ \mathrm{E}_1[V_1]$, so we conclude by Lemma B.4 and Clause 7. Therefore $\bar{s}.t \preceq_F low$, and since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. By Lemma B.4 and Clause 8 we can conclude.

□

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

Proposition B.15 (Strong Bisimulation for Low-Termination).
If we have $M_1\ \mathcal{T}^{m_j}_{F,low}\ M_2$ and also $W \vdash \langle\{{M_1}^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{{M_1'}^{m_j}\}, T_1', S_1'\rangle$, with $\langle T_1, S_1\rangle =^{F\cup F',low} \langle T_2, S_2\rangle$ such that n is fresh for T_2 if $\bar{n}_{\bar{k}} \in \mathrm{dom}(T_1' - T_1)$ and a is fresh for S_2 if $a_{l,\theta} \in \mathrm{dom}(S_1' - S_1)$, then there exist T_2', M_2' and S_2' such that $W \vdash \langle\{{M_2}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{{M_2'}^{m_j}\}, T_2', S_2'\rangle$ with $M_1'\ \mathcal{T}^{m_j}_{F,low}\ M_2'$ and $\langle T_1', S_1'\rangle =^{F,low} \langle T_2', S_2'\rangle$.

Proof. In the following, we use Subject Reduction (Theorem B.5) 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 m_j, low and F, is a requirement for being in the $\mathcal{T}^{m_j}_{F,low}$ relation.

- [Clause 1.] This case is not possible.
- [Clause 2.] Here $M_1 = M_2$. By Guaranteed Transitions (Lemma B.11) there exist T_2', M_2' and S_2' such that $W \vdash \langle\{M_2^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{M_2'^{m_j}\}, T_2', S_2'\rangle$ with $\langle T_1', S_1'\rangle =^{F\cup F', low} \langle T_2', S_2'\rangle$.
  - [$\boldsymbol{M_2' = M_1'}$.] Then we have $M_1' \; \mathcal{T}^{m_j}_{F,low} \; M_2'$, by Clause 2 and Subject Reduction (Theorem B.5).
  - [$\boldsymbol{M_2' \neq M_1'}$.] Then by Splitting Computations (Lemma B.6) ($N^{\bar{n}_{\bar{k}}} = ()$) and we have two possibilities:
    (1) there exist E and $a_{l,\theta}$ such that $M_1' = \mathrm{E}[S_1(a_{l,\theta})]$, $F' = \lceil \mathrm{E} \rceil$, $M_2' = \mathrm{E}[S_2(a_{l,\theta})]$, $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$ and $\langle T_2', S_2'\rangle = \langle T_2, S_2\rangle$. Since $S_1(a_{l,\theta}) \neq S_2(a_{l,\theta})$, we have $l \not\preceq_{F\cup F'} low$. Therefore, $M_1' \; \mathcal{T}^{m_j}_{F,low} \; M_2'$, by Lemma B.14 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 $T_1(m_j) \neq T_2(m_j)$ 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 $T_1(m_j) \neq T_2(m_j)$, we have $j \not\preceq_F low$, and by Lemma B.7 we have $s.t \not\preceq_F low$, which contradicts the assumption.
- [Clause 3.] Here $M_1 = (\bar{M}_1; \bar{N})$ and $M_2 = (\bar{M}_2; \bar{N})$ where $\bar{M}_1 \; \mathcal{T}^{m_j}_{F,low} \; \bar{M}_2$. Then either:
  - [$\boldsymbol{\bar{M}_1}$ can compute.] In this case we have $M_1' = (\bar{M}_1'; \bar{N})$ with $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Clause 3 and Subject Reduction (Theorem B.5) to conclude.
  - [$\boldsymbol{\bar{M}_1}$ is a value.] In this case $M_1' = \bar{N}$ and $F' = \emptyset$, $N^{\bar{n}_{\bar{k}}} = ()$ and $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$. We have $\bar{M}_2 \in$ ***Val*** by Remark B.13, hence $W \vdash \langle\{M_2{}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{N}^{m_j}\}, T_2, S_2\rangle$, and we conclude using Clause 2 and Subject Reduction (Theorem B.5).
- [Clause 4.] Here $M_1 = (\mathrm{ref}_{l,\theta} \; \bar{M}_1)$ and $M_2 = (\mathrm{ref}_{l,\theta} \; \bar{M}_2)$ where $\bar{M}_1 \; \mathcal{T}^{m_j}_{F,low} \; \bar{M}_2$, and $l \not\preceq_F low$. There are two cases.
  - [$\boldsymbol{\bar{M}_1}$ can compute.] In this case we have $M_1' = (\mathrm{ref}_{l,\theta} \; \bar{M}_1)$ with $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Subject Reduction (Theorem B.5) and Clause 4 to conclude.
  - [$\boldsymbol{\bar{M}_1}$ is a value.] In this case $M_1' = a_{l,\theta}$, with a fresh for S_1, $F' = \emptyset$, $N^{\bar{n}_{\bar{k}}} = ()$ and $\langle T_1', S_1'\rangle = \langle T_2, S_1 \cup \{a_{l,\theta} \mapsto \bar{M}_1\}\rangle$ (and therefore a is also fresh for S_2). Then $\bar{M}_2 \in$ ***Val*** by Remark B.13, and therefore $W \vdash \langle\{M_2{}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{a_{l,\theta}{}^{m_j}\}, T_2', S_2 \cup \{a_{l,\theta} \mapsto \bar{M}_2\}\rangle$. If we let $S_2' = S_2 \cup \{a_{l,\theta} \mapsto \bar{M}_2\}$ then $\langle T_1', S_1'\rangle =^{F,low} \langle T_2', S_2'\rangle$ since $l \not\preceq_F low$. We conclude using Clause 1 and Subject Reduction (Theorem B.5).
- [Clause 5.] Here $M_1 = (! \; \bar{M}_1)$ and $M_2 = (! \; \bar{M}_2)$ where $\bar{M}_1 \; \mathcal{T}^{m_j}_{F,low} \; \bar{M}_2$. We distinguish two sub-cases.
  - [$\boldsymbol{\bar{M}_1}$ can compute.] In this case $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Subject Reduction (Theorem B.5) and Clause 5 to conclude.
  - [$\boldsymbol{\bar{M}_1}$ is a value.] Then $\bar{M}_1 = a_{l,\theta}$ and $M_1' \in$ ***Val***, $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$, $F' = \emptyset$ and $N^{\bar{n}_{\bar{k}}} = ()$. By Remark B.13, $\bar{M}_2 \in$ ***Val***, and since M_1 and M_2 have the same type, it must be a reference $a_{l',\theta}$. Then, $W \vdash \langle\{M_2{}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{M_2{}'^{m_j}\}, T_2, S_2\rangle$ with $M_2{}' \in$ ***Val***, and we conclude using Clause 1 and Subject Reduction (Theorem B.5).
- [Clause 6.] Here we have $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ where $\bar{M}_1 \; \mathcal{T}^{m_j}_{F,low} \; \bar{M}_2$ and $\bar{N}_1 \; \mathcal{T}^{m_j}_{F,low} \; \bar{N}_2$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta \; \mathrm{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$. We distinguish three sub-cases.
  - [$\boldsymbol{\bar{M}_1}$ can compute.] In this case $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Subject Reduction (Theorem B.5) and Clause 6 to conclude.
  - [$\boldsymbol{\bar{M}_1}$ is value, but $\boldsymbol{\bar{N}_1}$ can compute.] In this case we have that $W \vdash \langle\{\bar{N}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{N}_1'^{m_j}\}, T_1', S_1'\rangle$. By Remark B.13 also $\bar{M}_2 \in$ ***Val***. We use the induction hypothesis, Subject Reduction (Theorem B.5) and Clause 6 to conclude.
  - [$\boldsymbol{\bar{M}_1}$ and $\boldsymbol{\bar{M}_1}$ are values.] Then $\bar{M}_1 = a_{l,\theta}$ and $M_1' = ()$, $\langle T_1', S_1'\rangle = \langle T_1, \{V \mapsto \bar{M}_1\}S_1\rangle$, $F' = \emptyset$ and $N^{\bar{n}_{\bar{k}}} = ()$. By Remark B.13, also $\bar{M}_2, \bar{N}_2 \in$ ***Val***, and since $\bar{M}_1$ and $\bar{M}_2$ have the same type, $\bar{M}_2$ must be a reference $a_{l',\theta'}$. Then, $W \vdash \langle\{M_2{}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{M_2{}'^{m_j}\}, T_2, \{V' \mapsto \bar{M}_2\}S_2\rangle$ with $\bar{M}_2' \in$

Val. Since $l \not\preceq_F low$, then we know that $\{V \mapsto \bar{M}_1\}S_1 =^{F\cup F',low} \{V' \mapsto \bar{M}_2\}S_2$. We conclude using Clause 1 and Subject Reduction (Theorem B.5).

- [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 $\bar{M}_1 \; \mathcal{T}^{m_j}_{F\cup\bar{F},low} \; \bar{M}_2$. There are two cases.
  - [$\bar{M}_1$ **can compute.**] In this case $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F'']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$ with $F' = \bar{F} \cup F''$. By induction hypothesis, we have $W \vdash \langle\{\bar{M}_2^{m_j}\}, T_2, S_2\rangle \xrightarrow[F'']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_2'^{m_j}\}, T_2', S_2'\rangle$, and $M_1' \; \mathcal{T}^{m_j}_{F\cup\bar{F},low} \; M_2'$ and $\langle T_1', S_1'\rangle =^{F\cup\bar{F},low} \langle T_2', S_2'\rangle$. Notice that $\langle T_1', S_1'\rangle =^{F,low} \langle T_2', S_2'\rangle$. We use Subject Reduction (Theorem B.5) and Clause 7 to conclude.
  - [$\bar{M}_1$ **is a value.**] In this case $M_1' = \bar{M}_1$, $F' = \emptyset$, $N^{\bar{n}_{\bar{k}}} = ()$ and $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$. Then $\bar{M}_2 \in$ ***Val*** by Remark B.13, and so $W \vdash \langle\{M_2{}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{\bar{n}_{\bar{k}}}} \langle\{\bar{M}_2^{m_j}\}, T_2, S_2\rangle$. We conclude using Clause 1 and Subject Reduction (Theorem B.5).

$\square$

We have seen in Remark B.13 that when two expressions are related by $\mathcal{T}^j_{F,low}$ 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 $\mathcal{T}^j_{F,low}$ 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 B.16 (Derivative of an Expression)**.** *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 W, F, N^{n_k}:*

$$W \vdash \langle M'', T_1, S_1\rangle \xrightarrow[F]{N^{n_k}} \langle M', T_1', S_1'\rangle$$

Definition B.17 (Non-resolvable Expressions)**.** *We say that an expression M is* non-resolvable, *denoted $M\dagger$, if there is no derivative M' of M such that $M' \in$* ***Val***.

Lemma B.18. *If for some F, low and j we have that $M \; \mathcal{T}^j_{F,low} \; N$ and $M\dagger$, then $N\dagger$.*

Proof. Let us suppose that $\neg N\dagger$. That means that there exists a finite number of states $\langle T_1, S_1\rangle, \ldots, \langle T_n, S_n\rangle$, and $\langle T_1', S_1'\rangle, \ldots, \langle T_n', S_n'\rangle$ and of expressions $N_1, \ldots, N_n$ such that

$$\begin{array}{rcl} W \vdash \langle\{N\}, T_1, S_1\rangle & \rightarrow & \langle\{N_1\}, T_1', S_1'\rangle \text{ and} \\ W \vdash \langle\{N_1\}, T_2, S_2\rangle & \rightarrow & \langle\{N_2\}, T_2', S_2'\rangle \text{ and} \\ & \vdots & \\ W \vdash \langle\{N_{n-1}\}, T_n, S_n\rangle & \rightarrow & \langle\{N_n\}, T_n', S_n'\rangle \end{array}$$

and such that $N_n \in$ ***Val***. By Strong Bisimulation for Low-Terminating Threads (Proposition B.15), we have that there exists a finite number of states $\langle\bar{T}_1', \bar{S}_1'\rangle, \ldots, \langle\bar{T}_n', \bar{S}_n'\rangle$ and of expressions $\bar{M}_1, \ldots, \bar{M}_n$ such that

$$\begin{array}{rcl} W \vdash \langle\{M\}, T_1, S_1\rangle & \rightarrow & \langle\{M_1\}, \bar{T}_1', \bar{S}_1'\rangle \text{ and} \\ W \vdash \langle\{M_1\}, T_2, S_2\rangle & \rightarrow & \langle\{M_2\}, \bar{T}_2', \bar{S}_2'\rangle \text{ and} \\ & \vdots & \\ W \vdash \langle\{M_{n-1}\}, T_n, S_n\rangle & \rightarrow & \langle\{M_n\}, \bar{T}_n', \bar{S}_n'\rangle \end{array}$$

such that:

$$M_1 \; \mathcal{T}^j_{F,low} \; \bar{N}_1, \textit{ and } \ldots, \textit{ and } M_n \; \mathcal{T}^j_{F,low} \; \bar{N}_n$$

By Remark B.13, we then have that $M_n \in$ ***Val***. Since M_n is a derivative of M, we conclude that $\neg M\dagger$. $\square$

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

Lemma B.19 (Composition of High Expressions)**.** *Suppose that M^{m_j} is typable in F. Then:*

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

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

   then $M^{m_j} \in \mathcal{H}_{F,low}$.
5. *If $M = (M_1 := M_2)$ and M_1 has type $\theta \, \text{ref}_{l,n_k}$ with $l \not\preceq_F low$ and either*

- *$M_1\dagger$ and ${M_1}^{m_j} \in \mathcal{H}_{F,low}$, or*
- *${M_1}^{m_j}, {M_2}^{m_j} \in \mathcal{H}_{F,low}$,*

then $M^{m_j} \in \mathcal{H}_{F,low}$.

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

Proof. We give the proof for the case where $M = (M_1\ M_2)$ and M_1 is a syntactically (F, low, j)-high function. The other cases are analogous or simpler.

- **[$M_1\dagger$ and ${M_1}^{m_j} \in \mathcal{H}_{F,low}$.]** Let $\mathcal{F}$ be a set of threads that includes $\mathcal{H}_{F,low}$, and that contains the threads $(M_1\ M_2)^{m_j}$ provided that they are typable in F, and satisfy $M_1 \notin$ ***Val*** and $M_1^{m_j} \in \mathcal{F}$ and M_1 is a (F, low, j)-high function. Assume that an application $M = (M_1\ M_2)$ such that $M_1\dagger$ and ${M_1}^{m_j} \in \mathcal{H}_{F,low}$ performs the transition $W \vdash \langle M^{m_j}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle M'^{m_j}, T', S'\rangle$. We show that this implies $M'^{m_j}, N^{n_k} \in \mathcal{F}$ and $\langle T', S'\rangle =^{F,low} \langle T', S'\rangle$.

  Since M_1 is non-resolvable, M_1 cannot be a value, and since M can compute, then also M_1 can compute. We then have $M' = (M_1'\ M_2)$ with $W \vdash \langle {M_1}^{m_j}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle M_1'^{m_j}, T', S'\rangle$. Since ${M_1}^{m_j} \in \mathcal{H}_{F,low}$, then also $M_1'^{m_j}, N^{n_k} \in \mathcal{H}_{F,low}$, thus $M_1'^{m_j}, N^{n_k} \in \mathcal{F}$, and $\langle T', S'\rangle =^{F,low} \langle T', S'\rangle$. By Subject Reduction (Theorem B.5), M_1' is a (F, low)-high function, and since $M_1\dagger$ then $M_1' \notin$ ***Val***. Hence $M'^{m_j} \in \mathcal{F}$.

- **[${M_1}^{m_j}, {M_2}^{m_j} \in \mathcal{H}_{F,low}$.]** Let $\mathcal{F}$ be a set of pools of threads that includes $\mathcal{H}_{F,low}$, and that contains threads $(M_1\ M_2)^{m_j}$ provided they are typable in F and satisfy $M_1^{m_j}, M_2^{m_j} \in \mathcal{F}$ and M_1 is a (F, low, j)-high function. Assume that such an application $M = (M_1\ M_2)$ performs the transition $W \vdash \langle M^{m_j}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle M'^{m_j}, T', S'\rangle$. We show that this implies $M'^{m_j}, N^{n_k} \in \mathcal{F}$ and $\langle T', S'\rangle =^{F,low} \langle T', S'\rangle$.

  - [$\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ are values.] Then $M_1 = (\lambda x.\bar{M}_1)$, $M' = \{x \mapsto M_2\}\bar{M}_1$ and $N' = ()$, $\langle T', S'\rangle = \langle T, S\rangle$. Since M_1 is a (F, low, j)-high function, then by Abs $\bar{M}_1$ is syntactically (F, low, j)-high, and by Substitution (Lemma B.3), also M' is syntactically (F, low, j)-high. Therefore, by High Expressions (Lemma B.10), $M'^{m_j} \in \mathcal{H}_{F,low}$.
  - [$\boldsymbol{M_1}$ can compute.] Then $M' = (M_1'\ M_2)$ with $W \vdash \langle {M_1}^{m_j}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle {M_1'}^{m_j}, T', S'\rangle$. Since ${M_1}^{m_j} \in \mathcal{H}_{F,low}$, then also $M_1'^{m_j}, N^{n_k} \in \mathcal{F}$ and $\langle T', S'\rangle =^{F,low} \langle T', S'\rangle$. By Subject Reduction (Theorem B.5) M_1' is a (F, low)-high function. Hence $M' \in \mathcal{F}$.
  - [$\boldsymbol{M_1}$ is a value but $\boldsymbol{M_2}$ can compute.] Then we have $M' = (M_1\ M_2')$ with $W \vdash \langle {M_2}^{m_j}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle M_2'^{m_j}, T', S'\rangle$. Since ${M_2}^{m_j}, N^{n_k} \in \mathcal{H}_{F,low}$, then also $M_2'^{m_j}, N^{n_k} \in \mathcal{F}$ and $\langle T', S'\rangle =^{F,low} \langle T', S'\rangle$. Hence $M' \in \mathcal{F}$.

□

Lemma B.20. *If for some j, F and low we have that $M_1\ \mathcal{T}^j_{F,low}\ M_2$ and $M_1 \in \mathcal{H}_{F,low}$, then $M_2 \in \mathcal{H}_{F,low}$.*

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

- [Clause 1.] Direct.
- [Clause 2.] Direct.
- [Clause 3.] Here $M_1 = (\bar{M}_1; \bar{N})$ and $M_2 = (\bar{M}_2; \bar{N})$ with $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$. Clearly we have that $\bar{M}_1 \in \mathcal{H}_{F,low}$, so by induction hypothesis, also $\bar{M}_2 \in \mathcal{H}_{F,low}$. We distinguish two sub-cases:
  - [$\bar{\boldsymbol{N}} \in \mathcal{H}_{F,low}$.] Then, $\bar{M}_2, \bar{N} \in \mathcal{H}_{F,low}$. Therefore, by Composition of High Expressions (Lemma B.19) we have that $M_2 \in \mathcal{H}_{F,low}$.
  - [$\bar{\boldsymbol{N}} \notin \mathcal{H}_{F,low}$.] Then $\bar{M}_1\dagger$, and by Lemma B.18 also $\bar{M}_2\dagger$. Therefore, by Composition of High Expressions (Lemma B.19) we have that $M_2 \in \mathcal{H}_{F,low}$.
- [Clause 4.] Here $M_1 = (\text{ref}_{l,\theta}\ \bar{M}_1)$ and $M_2 = (\text{ref}_{l,\theta}\ \bar{M}_2)$ where $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $l \not\preceq_F low$. Clearly we have that $\bar{M}_1 \in \mathcal{H}_{F,low}$, so by induction hypothesis also $\bar{M}_2 \in \mathcal{H}_{F,low}$. Therefore, by Composition of High Expressions (Lemma B.19) we have that $M_2 \in \mathcal{H}_{F,low}$.
- [Clause 5.] Here $M_1 = (!\ \bar{M}_1)$ and $M_2 = (!\ \bar{M}_2)$ where $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$. Clearly we have that $\bar{M}_1 \in \mathcal{H}_{F,low}$, so by induction hypothesis also $\bar{M}_2 \in \mathcal{H}_{F,low}$. This implies that $\bar{M}_2 \in \mathcal{H}_{F,low}$.
- [Clause 6.] Here we have $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ where $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta\ \text{ref}_l$ for some θ and l such that $l \not\preceq_F low$, and $\bar{N}_1\ \mathcal{T}^j_{F,low}\ \bar{N}_2$. Clearly we have that $\bar{M}_1 \in \mathcal{H}_{F,low}$, so by induction hypothesis also $\bar{M}_2 \in \mathcal{H}_{F,low}$. We distinguish two sub-cases:

- $[\bar{N}_2 \in \mathcal{H}_{F,low}.]$ Then, $\bar{M}_2, \bar{N}_2 \in \mathcal{H}_{F,low}$ where $\bar{M}_2$ has type $\theta\,\mathrm{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$. Therefore, by Composition of High Expressions (Lemma B.19) we have that $M_2 \in \mathcal{H}_{F,low}$.
- $[\bar{N}_2 \notin \mathcal{H}_{F,low}.]$ Then $\bar{M}_1\dagger$, and by Lemma B.18 also $\bar{M}_2\dagger$. Therefore, since $\bar{M}_2$ has type $\theta\,\mathrm{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$, by Composition of High Expressions (Lemma B.19) we have that $M_2 \in \mathcal{H}_{F,low}$.

- [Clause 7.] Here we have $M_1 = (\mathrm{flow}\ F'\ \mathrm{in}\ \bar{M}_1)$ and $M_2 = (\mathrm{flow}\ F'\ \mathrm{in}\ \bar{M}_2)$ with $\bar{M}_1\ \mathcal{T}^j_{F\cup F',low}\ \bar{M}_2$. Clearly we have that $\bar{M}_1 \in \mathcal{H}_{F,low}$, so by induction hypothesis also $\bar{M}_2 \in \mathcal{H}_{F,low}$. Therefore, by Composition of High Expressions (Lemma B.19) we have that $M_2 \in \mathcal{H}_{F,low}$.

□

B.2.3 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 sub-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 B.8 and B.9).

Lemma B.21 (High Threads might Split)**.** *Consider a thread M^{m_j} for which there exist Γ, F, s and τ such that $\Gamma \vdash^j_F M : s, \tau$ and suppose that $M = \mathrm{E}[(\mathrm{allowed}\ F'\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f)]$ with $j \not\preceq_F low$. Then $M^{m_j} \in \mathcal{H}_{F,low}$.*

Proof. By induction on the structure of E, using Lemma B.7 and Lemma B.10 and Lemma B.7. Consider $M = \mathrm{E}[M_0]$, where $M_0 = (\mathrm{allowed}\ F'\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f)$.

- $[\mathbf{E}[M_0] = M_0.]$ Then, using rule ALLOW, we have that $\Gamma \vdash^j_F (\mathrm{allowed}\ F'\ \mathrm{then}\ N_t\ \mathrm{else}\ N_f) : s, \tau$ where $\Gamma \vdash^j_F N_t : s_t, \tau$, $\Gamma \vdash^j_F N_f : s_f, \tau$ and $j \preceq_F s_t.w, s_f.w$. This means $s_t.w, s_f.w \not\preceq_F low$, so by Lemma B.10, then ${N_t}^{m_j}, {N_f}^{m_j} \in \mathcal{H}_{F,low}$. By Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- $[\mathbf{E}[M_0] = (\mathbf{E_1}[M_0]\ M_1).]$ Then by rule APP we have that $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1, \tau_1 \xrightarrow[F,j]{s_1'} \sigma_1$ and $\Gamma \vdash^j_F M_1 : s_1'', \tau_1$ with $s_1.r \preceq_F s_1'.w$ and $s_1.r \preceq_F s_1''.w$. By Lemma B.7 we have $j \preceq s_1.t$, which implies that $j \preceq_F s_1.t$ and $s_1.t \not\preceq_F low$. 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 B.10) we have ${M_1}^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- $[\mathbf{E}[M_0] = (V\ \mathbf{E_1}[M_0]).]$ Then by APP we have $\Gamma \vdash^j_F V : s_1, \tau_1 \xrightarrow[F,j]{s_1'} \sigma_1$ and $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1'', \tau_1$ with $s_1''.t \preceq_F s_1'.w$ and $s_1'.t \preceq_F s_1''.w$. By Lemma B.7 we have $j \preceq s_1''.t$, which implies that $j \preceq_F s_1''.t$ and $s_1''.t \not\preceq_F low$, and so $s_1'.w \not\preceq_F low$. Therefore, $s_1'.w \not\preceq_F low$, and $s_1''.w \not\preceq_F low$, 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 $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- $[\mathbf{E}[M_0] = (\mathbf{if}\ \mathbf{E_1}[M_0]\ \mathbf{then}\ M_t\ \mathbf{else}\ M_f).]$ Then by rule COND we have that $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1, \mathsf{bool}$, and $\Gamma \vdash^j_F M_t : s_1', \tau_1$ and $\Gamma \vdash^j_F M_f : s_1'', \tau_1$ with $s_1.t \preceq_F s_1'.w, s_1'.w$. By Lemma B.7 we have $j \preceq s_1.t$, which implies that $j \preceq_F s_1.t$ and $s_1.t \not\preceq_F low$. Therefore, $s_1'.w, s_1'.w \not\preceq_F low$, so by High Expressions (Lemma B.10) we have ${M_t}^{m_j}, {M_t}^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- $[\mathbf{E}[M_0] = (\mathbf{E_1}[M_0]; M_1).]$ Then by SEQ we have that $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1, \tau_1$ and $\Gamma \vdash^j_F M_1 : s_1', \tau_1'$ with $s_1.t \preceq_F s_1'.w$. By Lemma B.7 we have $j \preceq s_1.t$, which implies that $j \preceq_F s_1.t$ and $s_1.t \not\preceq_F low$. Therefore, $s_1'.w \not\preceq_F low$, and by High Expressions (Lemma B.10) we have ${M_1}^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- $[\mathbf{E}[M_0] = (\mathbf{ref}_{l,\theta}\ \mathbf{E_1}[M_0]).]$ Then by REF we have that $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1, \theta$ with $s_1.t \preceq_F l$. By Lemma B.7 we have $j \preceq s_1.t$, which implies that $j \preceq_F s_1.t$ and $s_1.t \not\preceq_F low$. Therefore, $l \not\preceq_F low$, and by induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

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

- $[\mathbf{E}[M_0] = (\mathbf{E_1}[M_0] := M_1).]$ Then by ASS we have that $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1, \theta\,\mathrm{ref}_{\bar{l}}$ and $\Gamma \vdash^j_F M_1 : s_1', \tau_1$ with $s_1.t \preceq_F s_1'.w$ and $s_1.t \preceq_F \bar{l}$. By Lemma B.7 we have $j \preceq s_1.t$, which implies that $j \preceq_F s_1.t$ and $s_1.t \not\preceq_F low$. Therefore, $\bar{l} \not\preceq_F low$ and

$s'_1.w \not\preceq_F low$. Hence, by High Expressions (Lemma B.10) we have ${M_1}^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- **[$\mathbf{E}[M_0] = (V := \mathbf{E_1}[M_0])$.]** Then by ASS we have $\Gamma \vdash^j_F V : s_1, \theta\ \mathrm{ref}_{\bar{l},n_{\bar{k}}}$ and $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s'_1, \tau_1$ with $s'_1.t \preceq_F \bar{l}$. By Lemma B.7 we have $j \preceq s'_1.t$, which implies that $j \preceq_F s'_1.t$ and $s'_1.t \not\preceq_F low$. Therefore, $\bar{l} \not\preceq_F low$, and by induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

- **[$\mathbf{E}[M_0] = (\mathbf{flow}\ F'\ \mathbf{in}\ \mathbf{E_1}[M_0])$.]** Then by rule FLOW we have $\Gamma \vdash^j_{F\cup F'} \mathrm{E}_1[M_0] : s_1, \tau_1$. By induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F\cup F',low}$, which implies $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), we conclude that $M^{m_j} \in \mathcal{H}_{F,low}$.

- **[$\mathbf{E}[M_0] = (\mathbf{flow}\ \mathbf{E_1}[M_0]\ \mathbf{in}\ M_1)$.]** Then by FLOW we have that $\Gamma \vdash^j_F \mathrm{E}_1[M_0] : s_1, \mathsf{flow}_{\bar{F}}$ and $\Gamma \vdash^j_{F\cup\bar{F}} M_1 : s'_1, \tau_1$ with $s_1.t \preceq_F s'_1.w$. By Lemma B.7 we have $j \preceq s_1.t$, which implies that $j \preceq_F s_1.t$ and $s_1.t \not\preceq_F low$. Therefore, $s'_1.w \not\preceq_F low$, and by High Expressions (Lemma B.10) we have ${M_1}^{m_j} \in \mathcal{H}_{F\cup\bar{F},low}$. By induction hypothesis $\mathrm{E}_1[M_0]^{m_j} \in \mathcal{H}_{F,low}$. Then, by Composition of High Expressions (Lemma B.19), $M^{m_j} \in \mathcal{H}_{F,low}$.

□

We now design a binary relation on expressions that uses $\mathcal{T}^j_{F,low}$ to ensure that high-terminating expressions are always followed by operationally high ones. The definition of $\mathcal{R}^j_{G,F,low}$, abbreviated $\mathcal{R}^j_{F,low}$ when the global flow policy is G, is given in Figure 6. The flow policy F is assumed to contain G. Notice that it is a symmetric relation. In order to ensure that expressions that are related by $\mathcal{R}^j_{F,low}$ 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 B.23. *If $M_1\ \mathcal{T}^j_{F,low}\ M_2$, then $M_1\ \mathcal{R}^j_{F,low}\ M_2$.*

The above remark is used to prove the following lemma.

Lemma B.24. *If for some j, F and low we have that $M_1\ \mathcal{R}^j_{F,low}\ M_2$ and $M_1 \in \mathcal{H}_{F,low}$, then $M_2 \in \mathcal{H}_{F,low}$.*

Proof. By induction on the definition of $M_1\ \mathcal{R}^j_{F,low}\ M_2$, using Lemma B.20. □

We have seen in Splitting Computations (Lemma B.6) that two computations of the same expression can split only

---

Definition B.22 ($\mathcal{R}^j_{F,low}$). *We have that $M_1\ \mathcal{R}^j_{F,low}\ M_2$ if $\Gamma \vdash^j_F M_1 : s_1, \tau$ and $\Gamma \vdash^j_F M_2 : s_2, \tau$ for some Γ, s_1, s_2 and τ and one of the following holds:*

- *[Clause 1'.] ${M_1}^{m_j}, {M_2}^{m_j} \in \mathcal{H}_{F,low}$, or*
- *[Clause 2'.] $M_1 = M_2$, or*
- *[Clause 3'.] $M_1 = (\mathrm{if}\ \bar{M}_1\ \mathrm{then}\ \bar{N}_t\ \mathrm{else}\ \bar{N}_f)$ and $M_2 = (\mathrm{if}\ \bar{M}_2\ \mathrm{then}\ \bar{N}_t\ \mathrm{else}\ \bar{N}_f)$ with $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and ${\bar{N}_t}^{m_j}, {\bar{M}_f}^{m_j} \in \mathcal{H}_{F,low}$, or*
- *[Clause 4'.] $M_1 = (\bar{M}_1\ \bar{N}_1)$ and $M_2 = (\bar{M}_2\ \bar{N}_2)$ with $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and $\bar{N}_1^{m_j}, \bar{N}_2^{m_j} \in \mathcal{H}_{F,low}$, 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 $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $\bar{N}_1\ \mathcal{R}^j_{F,low}\ \bar{N}_2$, 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 $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and $\bar{N}^{m_j} \in \mathcal{H}_{F,low}$, or*
- *[Clause 7'.] $M_1 = (\bar{M}_1; \bar{N})$ and $M_2 = (\bar{M}_2; \bar{N})$ with $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, or*
- *[Clause 8'.] $M_1 = (\mathrm{ref}_{l,\theta}\ \bar{M}_1)$ and $M_2 = (\mathrm{ref}_{l,\theta}\ \bar{M}_2)$ with $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and $l \not\preceq_F low$, or*
- *[Clause 9'.] $M_1 = (!\ \bar{M}_1)$ and $M_2 = (!\ \bar{M}_2)$ with $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, or*
- *[Clause 10'.] $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ with $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and $\bar{N}_1^{m_j}, \bar{N}_2^{m_j} \in \mathcal{H}_{F,low}$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta\ \mathrm{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$, or*
- *[Clause 11'.] $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ with $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $\bar{N}_1\ \mathcal{R}^j_{F,low}\ \bar{N}_2$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta\ \mathrm{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$, or*
- *[Clause 12'.] $M_1 = (\mathrm{flow}\ F'\ \mathrm{in}\ \bar{M}_1)$ and $M_2 = (\mathrm{flow}\ F'\ \mathrm{in}\ \bar{M}_2)$ with $\bar{M}_1\ \mathcal{R}^j_{F\cup F',low}\ \bar{M}_2$.*

Figure 6. The relation $\mathcal{R}^j_{F,low}$

if the expression is about to read a reference that is given different values by the memories in which they compute. In Lemma B.25 we saw that the relation $\mathcal{T}^{j}_{F,low}$ 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 $\mathcal{R}^{j}_{F,low}$ relates the possible outcomes of typable expressions in general.

Lemma B.25. *If there exist Γ, A, s, τ such that $\Gamma \vdash^{j}_{F} \mathrm{E}[(!\ a_{l,\theta})] : s, \tau$ with $l \not\preceq_{F\cup\lceil \mathrm{E}\rceil} low$, then for any values $V_0, V_1 \in$ **Val** such that $\Gamma \vdash V_i : \theta$ we have $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$.*

Proof. By induction on the structure of E using Lemma B.4, Lemma B.14, Lemma B.10.

- **[$\mathbf{E}[(!\ a_{l,\theta})] = (!\ a_{l,\theta})$.]** We have $V_0\ \mathcal{R}^{j}_{F,low}\ V_1$ by Clause 1'.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{E_1}[(!\ a_{l,\theta})]\ M)$.]** By rule APP we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau} \xrightarrow[F,j]{\bar{s}'} \bar{\sigma}$ and $\Gamma \vdash^{j}_{F} M : \bar{s}'', \bar{\tau}$ with $\bar{s}.r \preceq_F \bar{s}'.w$ and $\bar{s}.t \preceq_F \bar{s}''.w$. By Lemma B.7, we have $l \preceq \bar{s}.r$. Therefore $l \preceq_F \bar{s}'.w$. Since by hypothesis $l \not\preceq_{F\cup\lceil \mathrm{E}_1\rceil} low$ (therefore $l \not\preceq_F low$), then $\bar{s}'.w \not\preceq_F low$, that is $\mathrm{E}_1[(!\ a_{l,\theta})]$ is a syntactically (F, low, j)-high function. By Lemma B.4, the same holds for $\mathrm{E}_1[V_0]$ and $\mathrm{E}_1[V_1]$. By induction hypothesis we conclude that $\mathrm{E}_1[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}_1[V_1]$.
  - [$\bar{s}.t \not\preceq_F low$.] Then $\bar{s}''.w \not\preceq_F low$ (and also $\bar{s}''.w \not\preceq low$) so by High Expressions (Lemma B.10) we have $M^{m_j} \in \mathcal{H}_{F,low}$. Therefore, we conclude $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$ by Clause 4' and Lemma B.4.
  - [$\bar{s}.t \preceq_F low$.] Then by Lemma B.14 we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Since $M\ \mathcal{R}^{j}_{F,low}\ M$ by Clause 2', we conclude that $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$ by Clause 5' and Lemma B.4.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (V\ \mathbf{E_1}[(!\ a_{l,\theta})])$.]** By rule APP we have that $\Gamma \vdash^{j}_{F} V : \bar{s}, \bar{\tau} \xrightarrow[F,j]{\bar{s}'} \bar{\sigma}$ and $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}'', \bar{\tau}$ with $\bar{s}''.r \preceq_F \bar{s}'.w$. By Lemma B.7, we have $l \preceq \bar{s}''.r$, and so $l \preceq_F \bar{s}'.w$. Since by hypothesis $l \not\preceq_{F\cup\lceil \mathrm{E}_1\rceil} low$ (therefore $l \not\preceq_F low$), then $\bar{s}'.w \not\preceq_F low$, that is V is a syntactically (F, low, j)-high function. By Clause 1 we have $V\ \mathcal{T}^{j}_{F,low}\ V$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Therefore we conclude that $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$ by Clause 5' and Lemma B.4.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{if}\ \mathbf{E_1}[(!\ a_{l,\theta})]\ \mathbf{then}\ M_t\ \mathbf{else}\ M_f)$.]** By COND we have that $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \mathsf{bool}$, and $\Gamma \vdash^{j}_{F} M_t : \bar{s}_t, \bar{\tau}$ and $\Gamma \vdash^{j}_{F} M_f : \bar{s}_f, \bar{\tau}$ with $\bar{s}.r \preceq_F \bar{s}_t.w, \bar{s}_f.w$. By Lemma B.7, we have $l \preceq \bar{s}.r$ and so $l \preceq_F \bar{s}_t.w, \bar{s}_f.w$. Since by hypothesis $l \not\preceq_{F\cup\lceil \mathrm{E}_1\rceil} low$ (therefore $l \not\preceq_F low$), then $\bar{s}_t.w \not\preceq_F low$ and $\bar{s}_f.w \not\preceq_F low$. This implies that ${M_t}^{m_j}, {M_f}^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Therefore we conclude that $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$ by Clause 3' and Lemma B.4.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{E_1}[(!\ a_{l,\theta})]; M)$.]** By SEQ we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau}$ and $\Gamma \vdash^{j}_{F} M : \bar{s}', \bar{\tau}'$ with $\bar{s}.t \preceq_F \bar{s}'.w$.
  - [$\bar{s}.t \not\preceq_F low$.] Then $\bar{s}'.w \not\preceq_F low$ so by High Expressions (Lemma B.10) we have $M^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Then, $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$ by Clause 6' and Lemma B.4.
  - [$\bar{s}.t \preceq_F low$.] Then by Lemma B.14 we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Therefore, we conclude using Clause 7' and Lemma B.4.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{ref}_{\bar{l},\bar{\theta}}\ \mathbf{E_1}[(!\ a_{l,\theta})])$.]** By REF we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau}$ with $\bar{s}.r = s.r \preceq_F \bar{l}$ and $\bar{s}.t = s.t$. Therefore, since $l \not\preceq_{F\cup\mathrm{E}} low$ implies $l \not\preceq_{F\cup\mathrm{E}_1} low$, then by induction hypothesis we have $\mathrm{E}_1[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. By Lemma B.7 we have $l \preceq s.r$, so $s.r \not\preceq_F low$. Therefore, $\bar{l} \not\preceq_F low$, and we conclude by Lemma B.4 and Clause 8'.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (!\ \mathbf{E_1}[(!\ a_{l,\theta})])$.]** By rule DER we have $\Gamma \vdash^{j}_{F} \mathrm{E}_1[(!\ a_{l,\theta})] : \bar{s}, \bar{\tau}$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. We conclude by Lemma B.4 and Clause 9'.
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (\mathbf{E_1}[(!\ a_{l,\theta})] := M)$.]** By rule ASS we have that $\Gamma \vdash^{j}_{F} \mathrm{E}_1[a_{l,\theta}] : \bar{s}, \bar{\theta}\ \mathrm{ref}_{\bar{l},\bar{n}_k}$ with $\bar{s}.r \preceq_F \bar{l}$ and $\bar{s}.t \preceq_F \bar{s}'.w$. By Lemma B.7 we have $l \preceq s.r$, so $s.r \not\preceq_F low$ and so $\bar{l} \not\preceq_F low$.
  - [$\bar{s}.t \not\preceq_F low$.] Then $\bar{s}'.w \not\preceq_F low$ so by High Expressions (Lemma B.10) we have $M^{m_j} \in \mathcal{H}_{F,low}$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Then, $\mathrm{E}[V_0]\ \mathcal{R}^{j}_{F,low}\ \mathrm{E}[V_1]$ by Clause 10' and Lemma B.4.
  - [$\bar{s}.t \preceq_F low$.] Then by Lemma B.14 we have $\mathrm{E}_1[V_0]\ \mathcal{T}^{j}_{F,low}\ \mathrm{E}_1[V_1]$. Therefore, we conclude using Lemma B.4, Clause 11' and Clause 2' (regarding M).
- **[$\mathbf{E}[(!\ a_{l,\theta})] = (V := \mathbf{E_1}[(!\ a_{l,\theta})])$.]** By rule ASS we have that $\Gamma \vdash^{j}_{F} V : \bar{s}, \bar{\theta}\ \mathrm{ref}_{\bar{l},\bar{n}_k}$, $\Gamma \vdash^{j}_{F} \mathrm{E}_1[a_{l,\theta}] : \bar{s}', \theta$ with $\bar{s}'.r \preceq_F \bar{l}$. By Lemma B.7 we have $l \preceq \bar{s}'.r$, so $l \preceq_F \bar{l}$. Then, we must have $\bar{l} \not\preceq_F low$,

since otherwise $l \preceq_{F \cup \mathrm{E}} low$. By Clause 1 we have that $V\ \mathcal{T}^j_{F,low}\ V$, and by induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{R}^j_{F,low}\ \mathrm{E}_1[V_1]$. We then conclude by Lemma B.4 and Clause 11'.

- **[E[(! $a_{l,\theta}$)] = (flow F' in E$_1$[(! $a_{l,\theta}$)]).]** By rule FLOW we have $\Gamma \vdash^j_{F \cup F'} V : s, \tau$. By induction hypothesis $\mathrm{E}_1[V_0]\ \mathcal{T}^j_{F \cup F', low}\ \mathrm{E}_1[V_1]$, so we conclude by Lemma B.4 and Clause 12'.

□

We now state a crucial result of the paper: the relation $\mathcal{R}^j_{F,low}$ is a sort of "strong bisimulation".

Proposition B.26 (Strong Bisimulation for Typable Low Threads)**.** *If $M_1\ \mathcal{R}^j_{F,low}\ M_2$ and $M_1 \notin \mathcal{H}_{F,low}$ and $W \vdash \langle\{{M_1}^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{{M'_1}^{m_j}\}, T'_1, S'_1\rangle$, with $\langle T_1, S_1\rangle =^{F \cup F', low} \langle T_2, S_2\rangle$ such that n is fresh for T_2 if $n \in \mathrm{dom}(T'_1 - T_1)$ and a is fresh for S_2 if $a_{l,\theta} \in \mathrm{dom}(S'_1 - S_1)$, then there exist T'_2, M'_2 and S'_2 such that $W \vdash \langle\{{M_2}^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{n_k}} \langle\{{M'_2}^{m_j}\}, T'_2, S'_2\rangle$ with $M'_1\ \mathcal{R}^j_{F,low}\ M'_2$ and $\langle T'_1, S'_1\rangle =^{F,low} \langle T'_2, S'_2\rangle$.*

Proof. We use Subject Reduction (Theorem B.5) to guarantee typability (with the same type) for m_j, low and F, which is a requirement for being in the $\mathcal{R}^j_{F,low}$ relation. We also use the Strong Bisimulation for Low Terminating Threads Lemma (Lemma B.15).

- [Clause 1'.] This case is excluded by assumption.
- [Clause 2'.] Here $M_1 = M_2$. By Guaranteed Transitions (Lemma B.11) there exist T'_2, M'_2 and S'_2 such that $W \vdash \langle\{M_2^{m_j}\}, T_2, S_2\rangle \xrightarrow[F']{N^{n_k}} \langle\{{M'_2}^{m_j}\}, T'_2, S'_2\rangle$ with $\langle T'_1, S'_1\rangle =^{F \cup F', low} \langle T'_2, S'_2\rangle$.
  - [$\boldsymbol{M'_2 = M'_1}$.] Then we have $M'_1\ \mathcal{R}^j_{F,low}\ M'_2$, by Clause 2' and Subject Reduction (Theorem B.5).
  - [$\boldsymbol{M'_2 \neq M'_1}$.] Then by Splitting Computations (Lemma B.6) we have that ($N^{n_k} = ()$) and we have two possibilities:
    (1) there exists E and $a_{l,\theta}$ such that $F' = \lceil \mathrm{E} \rceil$, $M'_1 = \mathrm{E}[S_1(a_{l,\theta})]$, $M'_2 = \mathrm{E}[S_2(a_{l,\theta})]$, $\langle T'_1, S'_1\rangle = \langle T_1, S_1\rangle$ and $\langle T'_2, S'_2\rangle = \langle T_2, S_2\rangle$. Since $S_1(a_{l,\theta}) \neq S_2(a_{l,\theta})$, we have $l \not\preceq_{F \cup F'} low$. Therefore, $M'_1\ \mathcal{R}^j_{F,low}\ M'_2$, by Lemma B.25 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 $T_1(m_j) \neq T_2(m_j)$ 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 $T_1(m_j) \neq T_2(m_j)$, we have $j \not\preceq_F low$, and by Lemma B.21 $M_1 \in \mathcal{H}_{F,low}$, which contradicts our assumption.
- [Clause 3'.] Here we have $M_1 = (\text{if } \bar{M}_1 \text{ then } \bar{M}_t \text{ else } \bar{M}_f)$ and $M_2 = (\text{if } \bar{M}_2 \text{ then } \bar{M}_t \text{ else } \bar{M}_f)$ with $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$ and $\bar{M}_t^{m_j}, \bar{M}_f^{m_j} \in \mathcal{H}_{F,low}$. We can assume that $\bar{M}_1^{m_j} \notin \mathcal{H}_{F,low}$, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). Therefore, $M'_1 = (\text{if } \bar{M}'_1 \text{ then } \bar{M}_t \text{ else } \bar{M}_f)$ with $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T'_1, S'_1\rangle$. We use the induction hypothesis, Clause 3' and Subject Reduction (Theorem B.5) to conclude.
- [Clause 4'.] Here $M_1 = (\bar{M}_1\, \bar{N}_1)$ and $M_2 = (\bar{M}_2\, \bar{N}_2)$ with $\bar{M}_1\, \mathcal{R}^j_{F,low}\, \bar{M}_2$, $\bar{M}_1$ and $\bar{M}_2$ are syntactically (F, low, j)-high functions, and $\bar{N}_1^{m_j}, \bar{N}_2^{m_j} \in \mathcal{H}_{F,low}$. We can assume that $\bar{M}_1$ can compute, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). Therefore, $M'_1 = (\bar{M}'_1\, \bar{N}_1)$ with $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T'_1, S'_1\rangle$. We use the induction hypothesis, Clause 4' and Subject Reduction (Theorem B.5) to conclude.
- [Clause 5'.] Here $M_1 = (\bar{M}_1\, \bar{N}_1)$ and $M_2 = (\bar{M}_2\, \bar{N}_2)$ with $\bar{M}_1\, \mathcal{T}^j_{F,low}\, \bar{M}_2$, $\bar{M}_1$ and $\bar{M}_2$ are syntactically (F, low, j)-high functions, and $\bar{N}_1\, \mathcal{R}^j_{F,low}\, \bar{N}_2$. We distinguish two sub-cases:
  - [$\boldsymbol{\bar{M}_1}$ can compute.] In this case there exists $\bar{M}'_1$ performing the transition $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T'_1, S'_1\rangle$. We use Lemma B.15, Subject Reduction (Theorem B.5) and Clause 5' to conclude.
  - [$\boldsymbol{\bar{M}_1}$ is a value.] Then by Remark B.13, $\bar{M}_2 \in$ ***Val***. We can assume that $\bar{N}_1^{m_j}, \bar{N}_2^{m_j} \notin \mathcal{H}_{F,low}$, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). Then, $\bar{N}_1$ can compute, and so there exist $\bar{N}'_1$ such that $W \vdash \langle\{\bar{N}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{N}_1'^{m_j}\}, T'_1, S'_1\rangle$ with $M'_1 = (\bar{M}_1\, \bar{N}'_1)$. We use the induction hypothesis, Clause 5' and Subject Reduction (Theorem B.5) to conclude.
- [Clause 6'.] Here $M_1 = (\bar{M}_1; \bar{N})$ and $M_2 = (\bar{M}_2; \bar{N})$ where $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$ and $\bar{N}^{m_j} \in \mathcal{H}_{F,low}$. We can assume that $\bar{M}_1^{m_j} \notin \mathcal{H}_{F,low}$, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). Therefore, we have $M'_1 = (\bar{M}'_1; \bar{N})$ with $W \vdash$

$\langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Clause 6' and Subject Reduction (Theorem B.5) to conclude.

- [Clause 7'.] Here $M_1 = (\bar{M}_1; \bar{N})$ and $M_2 = (\bar{M}_2; \bar{N})$ with $\bar{M}_1 \ \mathcal{T}^j_{F,low}\ \bar{M}_2$. We distinguish two sub-cases:
  - [$\bar{\boldsymbol{M}}_{\mathbf{1}}$ can compute.] In this case there exists $\bar{M}_1'$ performing the transition $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use Lemma B.15, Subject Reduction (Theorem B.5) and Clause 7' to conclude.
  - [$\bar{\boldsymbol{M}}_{\mathbf{1}}$ is a value.] Then $M_1' = \bar{N}$, $F = \emptyset$, $N^{n_k} = ()$ and $\langle T_1', S_1'\rangle = \langle T_1, S_1\rangle$. By Remark B.13, $\bar{M}_2 \in$ ***Val***. Then, we have $W \vdash \langle\{M_2^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{N}^{m_j}\}, T_1', S_1'\rangle$. We conclude using Lemma B.15 and Clause 2'.
- [Clause 8'.] Here $M_1 = (\text{ref}_{l,\theta}\ \bar{M}_1)$ and $M_2 = (\text{ref}_{l,\theta}\ \bar{M}_2)$ where $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and $l \not\preceq_F low$. We can assume that $\bar{M}_1^{m_j} \notin \mathcal{H}_{F,low}$, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). Then, $\bar{M}_1$ can compute, and $M_1' = (\text{ref}_{l,\theta}\ \bar{M}_1)$ with $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Subject Reduction (Theorem B.5) and Clause 8' to conclude.
- [Clause 9'.] Here $M_1 = (!\ \bar{M}_1)$ and $M_2 = (!\ \bar{M}_2)$ where $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$. We know that $\bar{M}_1$ can compute, since otherwise ${M_1}^{m_j} \in \mathcal{H}_{F,low}$. Then, we have $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Subject Reduction (Theorem B.5) and Clause 9' to conclude.
- [Clause 10'.] Here we have $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ where $\bar{M}_1\ \mathcal{R}^j_{F,low}\ \bar{M}_2$, and $\bar{N}_1^{m_j}, \bar{N}_2^{m_j} \in \mathcal{H}_{F,low}$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta\ \text{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$. We can assume that $\bar{M}_1$ can compute, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). Therefore, $M_1' = (\bar{M}_1' := \bar{N}_1)$ with $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use the induction hypothesis, Clause 10' and Subject Reduction (Theorem B.5) to conclude.
- [Clause 11'.] Here we have $M_1 = (\bar{M}_1 := \bar{N}_1)$ and $M_2 = (\bar{M}_2 := \bar{N}_2)$ where $\bar{M}_1\ \mathcal{T}^j_{F,low}\ \bar{M}_2$, and $\bar{M}_1, \bar{M}_2$ both have type $\theta\ \text{ref}_{l,n_k}$ for some θ and l such that $l \not\preceq_F low$, and $\bar{N}_1\ \mathcal{R}^j_{F,low}\ \bar{N}_2$. We can assume that M_1 cannot be a redex, with $\bar{M}_1, \bar{N}_1 \in$ ***Val***, since otherwise $M_1^{m_j} \in \mathcal{H}_{F,low}$ by Composition of High Expressions (Lemma B.19). There are two cases to consider:
  - [$\bar{\boldsymbol{M}}_{\mathbf{1}}$ can compute.] Then we have $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$. We use Lemma B.15, Clause 11' and Subject Reduction (Theorem B.5) to conclude.
  - [$\bar{\boldsymbol{M}}_{\mathbf{1}}$ is a value but $\bar{\boldsymbol{N}}_{\mathbf{1}}$ can compute.] Then by Remark B.13, $\bar{M}_2 \in$ ***Val***. Then we have $W \vdash \langle\{\bar{N}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F']{N^{n_k}} \langle\{\bar{N}_1'^{m_j}\}, T_1', S_1'\rangle$. We conclude using induction hypothesis, Clause 11' and Subject Reduction (Theorem B.5).
- [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 $\bar{M}_1\ \mathcal{R}^j_{F\cup F',low}\ \bar{M}_2$. We can assume that $\bar{M}_1^{m_j} \notin \mathcal{H}_{F\cup F',low}$, since otherwise $\bar{M}_1^{m_j} \notin \mathcal{H}_{F,low}$ and by Composition of High Expressions (Lemma B.19) $M_1^{m_j} \in \mathcal{H}_{F,low}$. Therefore $W \vdash \langle\{\bar{M}_1^{m_j}\}, T_1, S_1\rangle \xrightarrow[F'']{N^{n_k}} \langle\{\bar{M}_1'^{m_j}\}, T_1', S_1'\rangle$ with $F' = \bar{F} \cup F''$. By induction hypothesis, we have that $W \vdash \langle\{\bar{M}_2^{m_j}\}, T_2, S_2\rangle \xrightarrow[F'']{N^{n_k}} \langle\{\bar{M}_2'^{m_j}\}, T_2', S_2'\rangle$, and that $M_1'\ \mathcal{R}^j_{F\cup\bar{F},low}\ M_2'$ and also $\langle T_1', S_1'\rangle =^{F\cup\bar{F},low} \langle T_2', S_2'\rangle$. Notice that $\langle T_1', S_1'\rangle =^{F,low} \langle T_2', S_2'\rangle$. We use Subject Reduction (Theorem B.5) and Clause 12' to conclude.

□

B.2.4 Behavior of Sets of Typable Threads

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

Theorem B.27 (Soundness of Typing Non-disclosure for Networks.). *Consider a pool of threads P. If for all $M^{m_j} \in P$ there exist Γ, s and τ such that $\Gamma \vdash^j_\emptyset M : s, \tau$, then P satisfies the Non-disclosure for Networks policy.*

Proof. To conclude the proof of the Soundness Theorem, it remains to exhibit an appropriate bisimulation on pools of threads.

The definition of $\mathcal{R}^\star_{low}$ is inductively defined as follows:

$$a)\ \frac{M^{m_j} \in \mathcal{H}_{\emptyset,low}}{\{M^{m_j}\}\ \mathcal{R}^\star_{low}\ \emptyset} \qquad b)\ \frac{M^{m_j} \in \mathcal{H}_{\emptyset,low}}{\emptyset\ \mathcal{R}^\star_{low}\ \{M^{m_j}\}}$$

$$c)\ \frac{M_1\ \mathcal{R}^j_{\emptyset,low}\ M_2}{\{{M_1}^{m_j}\}\ \mathcal{R}^\star_{low}\ \{{M_2}^{m_j}\}} \qquad d)\ \frac{P_1\ \mathcal{R}^\star_{low}\ P_2 \quad Q_1\ \mathcal{R}^\star_{low}\ Q_2}{P_1 \cup Q_1\ \mathcal{R}^\star_{low}\ P_2 \cup Q_2}$$

One can check that the relation $\mathcal{R}^\star_{low}$ is a low-bisimulation, using Lemma B.24, Strong Bisimulation for Typable Low

Threads (Proposition B.26), Subject Reduction (Theorem B.5), and also Lemma B.2. The main result then follows easily.

□

C. Confinement analysis (proofs)

C.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 [?].

Remark C.1. *If $W \in$ **Pse** and $\Gamma \vdash W : s, \tau$, then for all flow policies F', we have that $\Gamma \vdash W : \bot, \tau$.*

Lemma C.2.

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

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

□

Lemma C.3 (Substitution)**.**
If $\Gamma, x : \sigma \vdash M : s, \tau$ and $\Gamma \vdash W : \sigma$ then $\Gamma \vdash \{x \mapsto W\}M : s, \tau$.

Proof. By induction on the inference of $\Gamma, x : \tau \vdash M : s, \sigma$, and by case analysis on the last rule used in this typing proof, using the previous lemma. □

Lemma C.4 (Replacement)**.**
If $\Gamma \vdash \mathrm{E}[M] : s, \tau$ is a valid judgment, then the proof gives M a typing $\Gamma \vdash M : \bar{s}, \bar{\tau}$ for some $\bar{s}$ and $\bar{\tau}$ such that $s \preceq \bar{s}$. In this case, if $\Gamma \vdash N : \bar{s}', \bar{\tau}$ with $\bar{s} \preceq \bar{s}'$, then $\Gamma \vdash \mathrm{E}[N] : s', \tau$, for some s' such that $s \preceq s'$.

Proof. By induction on the structure of E. □

Proposition C.5 (Subject Reduction)**.** *Consider a thread M^{m_j} for which there exist Γ, s and τ such that $\Gamma \vdash M : s, \tau$. Then, if $W \vdash \langle\{M^{m_j}\}, T, S\rangle \xrightarrow[F']{N^{n_k}} \langle\{M'^{m_j}\}, T', S'\rangle$, there exists s' such that $\Gamma \vdash M' : s', \tau$, and where $s \cup W(T(m_j)) \preceq s'$. Furthermore, $\exists s''$ such that $\Gamma \vdash N : s'', \mathsf{unit}$ and $s \cup W(T(n_k)) \preceq s''$.*

Proof. We consider the smallest $\bar{M}$ such that $M = \bar{\mathrm{E}}[\bar{M}]$ 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 $W \vdash \langle\{\hat{M}^{m_j}\}, T, S\rangle \xrightarrow[\hat{F}]{\hat{N}^{n_k}} \langle\{\hat{M}'^{m_j}\}, T', S'\rangle$. We then proceed by case analysis on the transition $W \vdash \langle\{\bar{M}^{m_j}\}, T, S\rangle \xrightarrow[\bar{F}]{\bar{N}^{n_k}} \langle\{\bar{M}'^{m_j}\}, T', S'\rangle$, using Lemmas C.3 and C.4.

Suppose that $M = \bar{\mathrm{E}}[\bar{M}]$ and that $W \vdash \langle\{\bar{M}^{m_j}\}, T, S\rangle \xrightarrow[\bar{F}]{\bar{N}^{n_k}} \langle\{\bar{M}'^{m_j}\}, T', \bar{S}'\rangle$. We start by observing that this implies $F' = \bar{F} \cup \lceil\bar{\mathrm{E}}\rceil$, $M' = \bar{\mathrm{E}}[\bar{M}']$, $\bar{N} = N$ 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 $W \vdash \langle\{\hat{M}^{m_j}\}, T, S\rangle \xrightarrow[\hat{F}]{\hat{N}^{n_k}} \langle\{\hat{M}'^{m_j}\}, T', S'\rangle$.

By Lemma C.4, we have $\Gamma \vdash \bar{M} : \bar{s}, \bar{\tau}$ in the proof of $\Gamma \vdash \bar{\mathrm{E}}[\bar{M}] : s, \tau$, for some $\bar{s}$ and $\bar{\tau}$. We proceed by case analysis on the transition $W \vdash \langle\{\bar{M}^{m_j}\}, T, S\rangle \xrightarrow[\bar{F}]{\bar{N}^{n_k}} \langle\{\bar{M}'^{m_j}\}, T', S'\rangle$, and prove that $\Gamma \vdash \bar{M}' : \bar{s}', \bar{\tau}$, for some $\bar{s}'$ such that $\bar{s} \cup W(T(m_j)) \preceq \bar{s}'$.

- **[$\bar{M} = ((\lambda x.\hat{M})\ V)$.]**
  Here we have $\bar{M}' = \{x \mapsto V\}\hat{M}$. By rule APP, we have $\Gamma \vdash (\lambda x.\hat{M}) : \hat{s}, \hat{\tau} \xrightarrow{\hat{s}'} \hat{\sigma}$ and $\Gamma \vdash V : \hat{s}'', \hat{\tau}$, where $\bar{s} \cup W(T(m_j)) \preceq \hat{s}'$. By ABS, then $\Gamma, x : \hat{\tau} \vdash \hat{M} : \hat{s}', \hat{\sigma}$, and by Remark C.1 we have $\Gamma \vdash V : \hat{\tau}$. Therefore, by Lemma C.3, we get $\Gamma \vdash \{x \mapsto V\}\hat{M} : \hat{s}', \hat{\sigma}$.

- **[$\bar{M} = (\text{if } tt \text{ then } N_t \text{ else } N_f)$.]**
  Here we have $\bar{M}' = N_t$. By COND, we have that $\Gamma \vdash N_t : s_t, \bar{\tau}$, where $\bar{s} \cup W(T(m_j)) \preceq s_t$.

- **[$\bar{M} = (\text{ref}_{l,\theta}\ V)$.]**
  Here we have $\bar{M}' = a_{l,\theta}$. By LOC, we have $\Gamma \vdash a : \top, \theta\ \text{ref}_l$, which satisfies $\bot \preceq s.r$, $s.w \preceq \top$, $\bot \preceq s.t$ and $s \cup W(T(m_j)) \preceq \top$.

- **[$\bar{M} = (!\ a_{l,\theta})$.]**
  Here we have $\bar{M}' = S(a_{l,\theta})$. By assumption, we have that $\Gamma \vdash S(a_{l,\theta}) : \top, \theta$, which satisfies $\bot \preceq s.r$, $s.w \preceq \top$ and $\bot \preceq s.t$ and $s \cup W(T(m_j)) \preceq \top$.

- **[$\bar{M} = (\text{flow } F' \text{ in } V)$.]**
  Here we have $\bar{M}' = V$. By rule FLOW, we have that $\Gamma \vdash V : \hat{s}', \tau$ and by Remark C.1, we have $\Gamma \vdash V : \top, \bar{\tau}$, which satisfies $\bot \preceq s.r$, $s.w \preceq \top$, $\bot \preceq s.t$ and $s \cup W(T(m_j)) \preceq \top$.

- **[$\bar{M} = (\text{allowed } F' \text{ then } N_t \text{ else } N_f)$ and $F' \subseteq W(T(m_j))^*$.]**
  Here we have $\bar{M}' = N_t$. By ALLOW, we have that $\Gamma \vdash N_t : s_t, \bar{\tau}$, where $\bar{s} = s \cup s_t - F' \cup s_f \cup W(T(m_j)) \preceq s_t$ holds because $s_t \subseteq (s_t - F' \cup W(T(m_j)))^* = (s_t \cup W(T(m_j)))^*$.

- **[$\bar{M} = (\text{allowed } F' \text{ then } N_t \text{ else } N_f)$ and $F' \not\subseteq W(T(m_j))^*$.]**

  Here we have $\bar{M}' = N_f$. By ALLOW, we have that $\Gamma \vdash N_f : s_f, \bar{\tau}$, where $\bar{s} \cup W(T(m_j)) \preceq s_t$.

The proof for the case $\bar{M} = (\varrho x.W)$ is analogous to the one for $\bar{M} = ((\lambda x.\hat{M})\ V)$, while the proofs for the cases $\bar{M} = (\text{if } f\!f \text{ then } N_t \text{ else } N_f)$ and $\bar{M} = (V; \hat{M})$ are analogous to the one for $\bar{M} = (\text{if } tt \text{ then } N_t \text{ else } N_f)$, and the ones for $\bar{M} = (a_{l,\theta} := V)$, $\bar{M} = (\text{thread}_l\ \hat{M})$ is analogous to the one for $\bar{M} = (\text{ref}_{l,\theta}\ V)$. By Lemma C.4, we can conclude that $\Gamma \vdash \bar{\text{E}}[\bar{M}'] : s', \tau$, for some s' such that $s \cup W(T(m_j)) \preceq s'$.

Now, if $N^{n_k} \neq ()$ (N^{n_k} is created), then $\exists \hat{N} : M = \bar{\text{E}}[(\text{thread}_k\ \hat{N})]$ and $\bar{N} = \hat{N}$. By Lemma C.4, we have $\Gamma \vdash (\text{thread}_k\ \hat{N}) : \hat{s}, \mathsf{unit}$ in the proof of $\Gamma \vdash \bar{\text{E}}[(\text{thread}\ \hat{N})] : s, \tau$, for some $\hat{s}$, and $\hat{\tau}$. By THR, we have $\Gamma \vdash \hat{N} : \hat{s}, \mathsf{unit}$. Therefore, $s \cup W(T(m_j)) \preceq \hat{s}$. □

C.2. Confinement for Networks

Proposition C.6 (Meaning of the declassification effect**).** *Consider a thread M^{m_j} for which there exist Γ, G, s and τ such that $\Gamma \vdash M : s, \tau$. If we have $W \vdash \langle\{M^{m_j}\}, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle\{M'^{m_j}\}, T', S'\rangle$, then $F \subseteq s$.*

Proof. We use induction on the inference of $\Gamma \vdash M : s, \sigma$, by case analysis on the last rule used in this typing proof (we show only the most interesting cases).

- [FLOW.]

  Here $M = (\text{flow } \bar{F} \text{ in } \bar{N})$, and we have $\Gamma \vdash \bar{N} : \bar{s}', \tau$ with $s = \bar{s} \curlyvee \bar{s}' \curlyvee \bar{F}$. There are two possibilities:

  - [$M = \bar{F}$ and $\bar{N}$ can compute.] We have $W \vdash \langle\{\bar{N}\}, T, S\rangle \xrightarrow[\bar{F}']{N^{n_k}} \langle\{\bar{N}'\}, T', S'\rangle$ with $F = \bar{F}' \cup \bar{F}$. By induction hypothesis, $\bar{F}' \subseteq \bar{s}'$. Since $\bar{s}' \subseteq s$, then $\bar{F} \subseteq s$, and since $\bar{F}' \subseteq s$, then $F \subseteq s$.
  - [$M = \bar{F}$ and $\bar{N} = V$.] Then we have $W \vdash \langle\{(\text{flow } \bar{F} \text{ in } \bar{N})\}, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle\{V\}, T, S\rangle$ with $F = \emptyset$, so $F \subseteq s$ holds vacuously.

- [ALLOW.]

  Here $M = (\text{allowed } \bar{F} \text{ then } N_t \text{ else } N_f)$ and we have $\Gamma \vdash N_t : \bar{s}_t, \tau$ and $\Gamma \vdash N_f : \bar{s}_f, \tau$ with $s = \bar{s} \curlyvee (s_t - \bar{F}) \curlyvee s_f$. There are two possibilities:

  - [$\bar{M} = \bar{F} \subseteq W(T(m_j))^*$.] Then we have $W \vdash \langle M, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle N_t, T, S\rangle$ with $F = \emptyset$, so $F \subseteq s^*$ holds vacuously.
  - [$\bar{M} = \bar{F} \not\subseteq A^*$.] Then we have $W \vdash \langle M, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle N_f, T, S\rangle$ with $F = \emptyset$, so $F \subseteq s^*$ holds vacuously.

□

Theorem C.7 (Soundness of Typing Confinement for Networks**).** *Consider a fixed policy-mapping W, a pool of threads P and its corresponding position tracker T, such that for all $M^{m_j} \in P$ there exist Γ, s and τ satisfying $\Gamma \vdash M : s, \tau$ and $W(T(m_j)) \preceq s$. Then the set* $\text{pair}(P, T)$ *is a set of operationally confined located threads.*

Proof. We show that the set

$$C = \{\langle d, M^{m_j}\rangle \mid \exists \Gamma, s, \tau \text{ such that } \Gamma \vdash M : s, \tau \text{ and } s \subseteq W(T(m_j))^*\}$$

is a set of operationally confined threads.

Consider a pair $\langle d, M^{m_j}\rangle \in C$, for which we have a transition $W \vdash \langle\{M^{m_j}\}, T, S\rangle \xrightarrow[F]{N^{n_k}} \langle\{M'^{m_j}\}, T', S'\rangle$. By Proposition C.6 we have that $F \subseteq s \subseteq W(T(m_j))$. By Subject Reduction (Proposition C.5) we have there exists s' such that $\Gamma \vdash M' : s', \tau$ and $s \cup W(T(m_j)) \preceq s'$, i.e. $s' \subseteq (s \cup (W(T(m_j)))^*$, and that there exists s'' such that $\Gamma \vdash N : s'', \mathsf{unit}$ and $s \cup W(T(n_k)) \preceq s''$. Note that if $N \neq ()$, then $T(m_j) = T'(n_k)$, so $s'' \subseteq W(T(m_j))^*$. It remains to prove that $s' \subseteq W(T'(m_j))^*$. The result is trivial when $T'(m_j) = T(m_j)$, so we will only check the case where $M = \text{E}[(\text{goto } d')]$. We then have that $T'(m_j) = d'$ and $s \subseteq W(d')$. By MIG we have $s = s'$, so we conclude that $s' \subseteq W(T'(m_j))$. □